前言

R软件是进行统计分析、绘图和统计编程的强大工具。现在成千上万的人用它来进行日常的重要统计分析。R软件是一个自由、开源的软件平台，它是许多聪明、勤奋工作的人们集体工作的成果。R软件有超过两千多个软件包插件。R软件是其他所有商业统计软件包的强劲竞争对手。

但是，开始使用R软件可能感到无从下手。对于许多任务，即便是一些基本的任务，R的实现也不是很明显。当了解了R的使用方法后，简单的问题自然能得心应手地解决，但学习“如何”使用R的过程有时会让人感到发狂。

本书介绍了如何使用R软件的一些方法，其中每一个方法对应解决某个特定的问题。介绍这些方法的途径是这样的：首先给出待解决的问题，然后给出解决方案的简单介绍，之后再给出对解决方案的讨论，深入剖析解决方案，给出该方案的原理。我知道这些方法有效实用，我也知道这些方法可行，因为我本人也使用它们。

这些方法所涉及的范围较为广泛。首先从基本的任务开始介绍，然后介绍数据的输入和输出、基础统计、绘图以及线性回归。与R有关的工作都将或多或少地涉及本书介绍的方法。

通过本书的讲解，初学者能快速地了解R并获得提高。如果你对R软件有一定的了解，那么本书也能帮助你巩固已学的知识，拓宽你的思维（例如，“下一次我应该怎么使用K-S检验”）。

从严格意义上来说，本书并不是一本关于R软件的教程，但你将会从中学习到许多R软件的应用技巧。本书也不是一本关于R的参考手册，但它确实包含了许多实用的内容。本书也不是一本R软件的编程指南，但书中很多方法都可以应用到R的编程脚本中。

最后，本书不是统计学理论的参考书。本书假设读者对统计理论和方法有一定的了解，他们需要知道的是如何在R软件中实现。

方法

本书介绍的大部分方法，都是由一两个R函数命令来解决某一特定问题。需要注意的是，书中不会对某一函数的全部功能进行详细解释，而是仅仅介绍那些为了解决某个问题所需要涉及的函数功能。R软件中几乎所有的函数都远远不止本书中所介绍的这些功能，其中有的函数具有更强大的功能。因此强烈建议读者阅读这些函数的帮助页面，你可能会从中得到不少收获。

每个方法都为读者提供了解决某个问题的一条途径。当然对于每个问题有可能存在多个正确的解决方案。在这种情况下，我一般会选择最为简单的方法介绍。对于书中给出的任何问题，你自己或许可以找到其他一些解决方案。本书着重介绍解决问题的方法，类似“菜谱”书，不是R软件的大全书籍。

尤其是，R软件有大量的可添加包，这几千个R软件包都可以通过网络下载的方式得到。这些包中含有许多替代算法和统计方法。本书侧重于R基础发布版所带的核心功能，因此你可以从其他的R添加包中找到相关的替代方案（参见方法1.11）。

对术语的说明

每个方法旨在迅速地解决问题，而非长篇大论地进行叙述。因此我可能会采用一些术语来简化相关内容的解释，这些术语有时候可能不精确，但是正确的。比如对于“泛型函数”一词。我把函数print(x)和函数plot(x)称为泛型函数，原因是它们能适当地处理多种输入参数x。计算机学家可能会质疑这一术语，因为严格地说这些都不是简单的“函数”，它们是多态方法并且动态调度。但是，如果我仔细地精确定义所有这样的技术细节，那么关键的解决方案将会埋没于这些细枝末节的技术问题中。所以为了便于阅读，我就将它们称为函数。

另一个例子是统计学中假设检验所用语义的严格性。若使用概率论的严格定义，就会使读者难以清晰理解这些检验的实际应用，所以我以更通俗的语言描述各个统计检验。更多有关假设检验方法的细节请查看第9章的简介。

我的目标是用通俗易懂而非严格的正式语言，让R软件能被更多的读者所理解和接受。因此希望各个领域的专家对于我所给出的某些并不严谨的术语定义予以谅解。

软件及平台说明

虽然R软件时常进行有计划的版本更新，但其语言定义和核心实现是稳定的。本书所介绍的方法将适用于基础发布版的任何最新版本。

有些方法对于操作平台有特殊的要求，我会在文中对其加以标注，这些方法大多数是一些有关软件本身的问题，如程序的安装和配置。据我所知，书中的其他所有方法都能在R的三个主要平台中得到兼容，即Windows、OS X和Linux / UNIX。

其他资源

网络

R项目网站（*http://www.r-project.org*）汇集了所有R软件的相关资源。从中可以下载R程序代码、R添加包、文档、源代码以及许多其他资源。

除了R项目网站以外，我建议使用一个针对R软件的搜索引擎，比如Sasha Goodman开发的Rseek搜索引擎（*http://rseek.org*）。也可以使用谷歌这样的通用搜索引擎，但在搜索“R”搜索词时可能会得到许多无关的搜索结果。更多有关网络搜索的细节参见方法1.10。

浏览博客也是学习R软件和掌握相关R最新动态的一种有效方式。网络中存在许多这样的博客，我推荐其中两个：Tal Galili的R-bloggers（*http://www.r-bloggers.com/*）和PlanetR的（*http://planetr.stderr.org*）。可以通过订阅他们的网站了解许多相关网站上有趣且实用的文章。

R软件参考书籍

市面上有许多学习和应用R软件的书籍。下面列出一些我发现会有帮助的R软件教程。R项目网站收录并编制了大量与R相关的书目（*http://www.r-project.org/doc/bib/R-books.html*）。我所推荐的书目有：

Network Theory Limited出版的《An Introduction to R》，作者是William Venables等。该书涵盖了许多对初学者很有帮助的知识。可以通过CRAN网站免费下载该书的PDF版本（*http://cran.r-project.org/doc/manuals/R-intro.pdf*），或者推荐购买纸质书籍，此书所获利润将捐赠给R项目。

O'Reilly公司出版的《R in a Nutshell》（*http://oreilly.com/catalog/978059680717*），该书的作者是Joseph Adler，此书可以随时作为你R软件的使用参考，并且它比本书涵盖了更多的内容。

任何应用R绘制正式图形的工作都可以参考《R Graphics》一书，该书的作者为Paul Murrel（Chapman & Hall/CRC）。根据应用的R图形包的不同，也可以参

考《Lattice:Multivaritate Data Visualiation with R》，作者为Deepayan Sarkar，由Springer出版社出版；《ggplot2: Elegant Graphics for Data Analysis》，作者为Hadley Wickham，由Springer公司出版。

《Modern Appied Statistic With S(4th ed.)》，作者为William Venables等，由Springer公司出版。该书采用S软件来说明一些高级的统计技术。该书所涉及的函数和数据可通过下载R软件标准发布版MASS获得。

市面上定期会有关于R编程的新书出版，但我并不提倡不加区分地选择。关于R软件编程，我推荐《R in a Nutshell》和William Venables与Brian Ripley等的《S Programming》（Springer）。我还推荐下载《R Language Definition》一书（*http://cran.r-project.org/doc/manuals/R-lang.pdf*），通过它可以解决许多R软件编程中遇到的细节问题。

统计学书籍

在你学习的过程中需要一本好的统计学参考书作为指导，它可以帮助你准确地理解在R中进行的统计检验。目前市面上有许多优秀的统计学参考书，因此我所推荐的书很难说这本比那本更优秀。

由John Verzani编写的《Using R for Introductory Statistic》（Chapman & Hall/CRC），是一本优秀的统计学教材。它结合统计学与R软件，讲述应用统计方法的一些必要的计算机技巧（*http://www.r-project.org/doc/bib/R-books.html*）。

越来越多的统计学作者选择R软件来讲述相应的统计方法。某一特定专业领域的工作者可以在R项目网站收录的书目中寻找所需要的书籍。

本书约定

在本书中使用以下排版方式：

斜体（*Italic*）

此类字体表示新出现的术语、网址、电子邮件地址、文件名、文件扩展名等。

等宽字体（`Constant width`）

此类字体用于程序清单，以及正文中出现的程序元素（如变量名或函数名、数据库名、数据类型、环境变量、程序语句、关键字等）。

等宽粗体（**`Constant width bold`**）

此类字体表示读者需要完整无误地输入该文本给出的指令或其他文字。

等宽斜体(*Constant width italic*)

此类字体表示读者应提供用户指定的值来替换这里的文字，或者根据上下文给出的值来替换这里的文本。

注意： 表示作者给出的提示、建议或标记内容。

警告： 表示读者阅读时要特别注意和慎重的内容。

代码示例的规定

本书内容旨在帮助读者完成工作。一般情况下，你都可以不受限制地在程序和文档中使用本书中的代码，除非你要复制本书中的大部分代码。例如，你可随意使用本书中的代码来编写程序，或者通过引用书中的案例和代码来解决一个问题，这都不需要获得许可。若将O'Reilly公司出版的书中的例子制成CD来销售或发行，或者引用书中大量的案例和代码作为自己产品的文档，则需要获得我们的许可。

我们希望但并不强制你在引用本书的内容时，说明引文的文献出处。一般标准的文献出处格式包括：标题，作者，出版商和ISBN号。例如：《R Cookbook》by Paul Teetor. Copyright 2011 Paul Teetor, 978-0-596-80915-7。

如果你对你所引用的本书代码和案例是否属于合理使用范围之内有所疑惑，可通过邮箱*permissions@oreilly.com*联系我们。

联系我们

有关本书的任何建议和疑问，可以通过下列方式与我们取得联系：

美国：

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472

中国：

北京市西城区西直门南大街2号成铭大厦C座807室（100035）
奥莱利技术咨询（北京）有限公司

要评论或询问本书的技术问题，请发送电子邮件到：

bookquestions@oreilly.com

有关我们的书籍、会议、资源中心以及O'Reilly网络，可以访问我们的网站：

http://www.oreilly.com
http://www.oreilly.com.cn

本书专设如下网站，我们会在其中公布书中的勘误、示例和其他附加信息。

http://www.oreilly.com/catalog/9780596809157。

对本书有任何疑问或意见，欢迎发送邮件到：

bookquestions@oreilly.com。

更多信息请访问我们的网站*http://oreilly.com*。

Facebook: *http://facebook.com/oreilly*。

Twitter: *http://twitter.com/oreillymedia*。

YouTube: *http://www.youtube.com/oreillymedia*。

致谢

我要对整个R社区，尤其是R软件的核心开发团队表示衷心感谢。他们的无私付出对世界统计学的贡献巨大。

我要感谢本书的技术审校者：James D. Long、Timothy McMurry、David Reiner、Jeffery Ryan和John Verzani。同时感谢Joe Adler给予本书的意见。他们做出的反馈对于本书得以有高质量、严谨并且实用的内容至关重要。他们的意见也帮助我节省了许多时间，避免了我传播错误的内容。

Mike Loukides是一位出色的编辑，我在此深深感谢他的智慧和指导。开始本书的项目时有人宣称Mike是出版行业里最棒的编辑，现在我完全相信这一事实。

我要对我的妻子Anna表达最大的谢意。她的支持使本书出版成为可能。她的参与使得编写本书的过程充满快乐。

第1章

R入门和获得帮助

简介

本章是其他章的基础，它介绍如何下载、安装和运行R软件。

本章重点说明通过何种途径寻找解决问题的方法。在R社区中提供了丰富的说明和帮助文件，因此你能从中得到他人的帮助。下面是一些常用的帮助来源：

本地已安装的文档

当在计算机上安装R软件时，它将自动安装大量的文档。可以对本地文档进行浏览（方法1.6）和搜索（方法1.8）。我经常在网上搜索答案，但常常惊奇地发现很多问题都能从本地已有的文档得以解答。

分类软件包视图

分类软件包视图（Task view）是对特定于某个统计工作领域的软件包的描述，例如计量经济学、医疗成像、心理学，或者空间统计。每个分类软件包视图都由该领域的专家来撰写和维护。CRAN网站上共有28个这样的软件包视图，所以你总能找到一个或者多个感兴趣的领域。我建议每个初学者试着阅读至少一个软件包视图，从而大致对R软件的功能有一个初步的了解（方法1.11）。

数据包文档

大多数R软件包都附带说明文档。许多R包甚至还包含对其内容的概括和教程，在R社区称为Vignette。这些文档能大大提高R软件包的使用效率。这些文档公布在包存储库中，如CRAN网站上。在安装R包的同时，其相关文档也会自动地安装到电脑上。

邮件列表

R软件的爱好者自发地在网站上通过邮件为初学者答疑，并将这些解答邮件以列表的形式公布在网站上，所以你可以通过查阅这些邮件来寻找答案（参见方法1.12）。

Q&A网站

可以通过问题与解答（Question and Answer，Q&A）网站进行提问，由知情人士自愿予以回复。网站可根据回答的质量进行投票，因此经长时间后可得到最优的解决方案。另外，所有的提问与解答都会标注、存档以供检索之用，此类网站是介于邮件列表和社区网站之间的一种形式，例如Stack Overflow site是一个优秀的Q&A网站。

其他网络

网络中有许多R软件相关的资料，可以通过一些R软件检索工具进行查询（参见方法1.10）。由于网络信息更新迅速，所以期望有更好的新途径来组织和寻找与R相关的帮助信息。

1.1 下载和安装R软件

问题

如何在计算机上安装R软件。

解决方案

Windows和OS X用户可以从CRAN网站下载R软件，Linux和UNIX用户则需要使用它们的软件包管理工具来安装R软件包。

Windows

1. 打开网址：*http://www.r-project.org/*。
2. 点击“CRAN”，你会看到一系列按照国家名称排序的镜像网站。
3. 选择与你所在地相近的网站。
4. 点击“Download and Install R”下的“Windows”。
5. 点击“base”。
6. 点击链接下载最新版本的R软件（后缀为*.exe*的文件）。
7. 下载完成后，双击程序文件（*.exe*文件）并回答安装中的常见问题，进行常规的安装操作。

OS X

1. 打开网址：*http://www.r-project.org/*。
2. 点击"CRAN"，你会看到一系列按照国家名称排序的镜像网站。
3. 选择与你所在地相近的网站。
4. 点击"MacOS X"。
5. 点击"Files:"下方的后缀为*.pkg*的文件，下载最新版本的R软件。
6. 下载完成后，双击已下载的后缀为*.pkg*的文件并回答安装中的常见问题，进行常规的安装操作。

Linux 或 UNIX

对于Linux的某些主要发行版本，它们有对应的R安装软件包。例如以下的发行版：

发行版本	R软件包名称
Ubuntu或Debian	r-base
Red Hat或Fedora	R.i386
Suse	R-base

通过操作系统的软件包管理器下载和安装R软件包。一般安装过程需要管理员密码或者sudo权限；否则，需要由系统管理员来进行安装。

讨论

由于R软件有Windows和OS X这两个平台的预编译二进制版本，所以安装过程很直接。你只要按照安装指令进行即可。CRAN网站也提供了有关安装的帮助资料，如安装时的常见问题和特殊情况下的窍门等一些有用的资料（例如，如何在Windows Vista环境下安装R软件）。

一般情况下，在Linux和UNIX上安装R软件有以下两种方法。安装发布版，或者从头进行编译安装。用发布版进行安装时，不管是初次安装还是后续更新都很方便，因此实际操作中推荐使用前一种方法。

在Ubuntu或Debian操作环境下，使用apt-get指令下载安装R软件。而运行apt-get程序需要必要的sudo权限，相应指令如下：

```
$ sudo apt-get install r-base
```

在Red Hat或Fedora操作环境下，使用yum软件包管理器，相应指令如下：

```
$ sudo yum install R.i386
```

目前大多数的操作平台都采用图形软件包管理器，使操作更为方便。

除了基础的R包以外，我建议同时下载安装文档包。比如我喜好查阅超链接文档，在我的Ubuntu计算机中添加了`r-base-html`和`r-doc-html`两个软件包。在Ubuntu平台上可以通过如下指令在本地计算机安装这两个重要的手册：

```
$ sudo apt-get install r-base-html r-doc-html
```

有的Linux软件库也包括在CRAN的预编译的R软件包，但我个人偏向于使用直接从CRAN上得到的官方R软件版本，它一般有最新版本。

如果UNIX版本不明或者该系统不支持，或者需要进行特殊的设置或者优化R软件，那么在这种情况下，可能需要从头进行编译安装R软件。在Linux和UNIX上编译安装流程很标准。从CRAN网站镜像中下载名为*R-2.12.1.tar.gz*的压缩包（其中的2.12.1可能会被最新版本号替代），解压缩后寻找名字为*INSTALL*的文件，按照指示进行安装。

另请参阅

对于下载安装，《R in a Nutshell》（O'Reilly）一书给出了更具体的说明，包括指示如何在Windows和OSX操作系统环境下进行手动安装。最系统地描述R软件在各个平台上如何进行安装的可能是名为“R安装和管理”（R Installation and Administration）的文件（*http://cran.r-project.org/doc/manuals/R-admin.html*），该文件可以通过CRAN网站下载得到，它讲述了在多种不同平台编译和安装R软件。

本方法是有关安装R基础发布版。如果安装CRAN中的添加包，另请参阅方法3.9。

1.2 开始运行R软件

问题

如何开始运行R软件。

解决方案

windows

点击“开始菜单”（Start）→选择“所有程序”（All Programs）→选择R，或者双击安装目录或桌面上的R软件图标（假定安装程序创建了一个R图标）。

OS X

点击*Applications*（应用程序）目录中的图标，或者把R图标放到停靠栏中，然后点击该R图标。也可以通过在UNIX命令行中输入“R”来运行R软件。

Linux和UNIX

通过输入R命令（大写的R）来运行R软件。

讨论

R软件运行方式视不同的操作平台而定。

在Windows中运行R

R软件在Windows中通过新建窗口运行，该窗口包含一个名为R控制台（R Console）的文本窗口，用户可通过该文本窗口输入R表达式（见图1-1）。

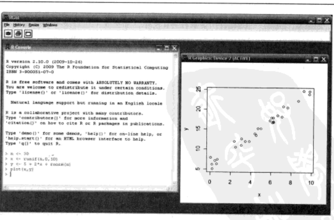


图1-1：Windows操作系统中运行R

R软件在Windows“开始”（Start）菜单中会出现一个奇怪的现象，即每当更新至新版本时系统会保留原有旧版本，因此有时会同时出现如“R 2.8.1”、“R 2.9.1”、“R 2.10.1”三个运行程序的情况。此情况下仅运行最新版本的R程序（也可以通过卸载旧版本避免这样的情况）。

通过“开始”（Start）菜单运行R软件较为繁琐，因此推荐在桌面上建立快捷方式，或者直接双击打开*.Rdata*文件运行R软件。

若R软件在安装过程中未添加桌面快捷方式，则可通过如下方式添加：选择“开始”（Start）菜单中的R程序文件，右键按住不放将其拖至桌面。系统会询问复制或剪切该图标，选择复制，完成快捷方式的添加。

也可以通过双击工作目录下的*.RData*文件来运行R程序，这个*.RData*文件是R软件创建并保存在工作目录下的一个文件。第一次运行R软件前，你可以先建立一个目录，然后启动R并把R的工作目录改变到刚建立的目录，即工作目录。在退出R时，可以把工作空间内容保存到该工作目录，也可以通过`save.image`函数来把工作空间内容保存到工作目录。工作空间内容保存后会得到一个后缀为*.RData*的文件。以后可以通过打开R工作目录，然后双击该目录下的*.RData*文件来启动R软件。

容易混淆的一点是，以不同的打开方式在Windows中打开R软件时，其工作目录也会不同。

- 如果从“开始”（Start）菜单打开R软件，那么其工作目录一般会设定为*C:\Documents and Settings\<username>\My Documents*（Windows XP）或*C:\Users\<username>\Documents*（Windows Vista、Windows 7）。也可以通过对环境变量R_USER的设定来改变目录路径。

- 如果通过桌面快捷方式打开R软件，在运行前可以手动设定R的另一个目录，它在R启动后将变为工作目录。具体操作如下：右击快捷方式，选择“属性”（Properties），在名为“起始位置”（Start in）一栏中输入指定的目录，点击“确定”（OK）按钮完成。

- 双击*.RData*文件来运行R软件是最直接的方法。R会自动将工作目录设定为该*.RData*文件所在的目录，因此以这种方式打开最为便利有效。

在任何情况下，都可以在R软件中使用`getwd`函数查询当前的工作路径（方法3.1）。

顺带一提，Windows版本的R软件中附带了另一个名为*Rterm.exe*的控制台程序，可以在安装路径下的*bin*子目录中找到该文件。相比R图形用户界面（Graphic User Interface，GUI）而言，*Rterm.exe*缺少图形用户界面（GUI），不便于使用。因此除非处理一些批量（非交互性）工作，如运行Windows计划程序安排的任务以外，其余情况不推荐使用该程序。本书假定读者使用R软件的GUI版本，而非控制台版本。

在OS X平台运行R

在*Applications*目录中点击R图标打开R软件的GUI版本（如果经常使用R，可将R图标从应用文件夹拖放到停靠栏以便使用），它比R的控制台版本方便。R软件的GUI版本显示工作空间目录，初始值为用户的主目录。

也可以通过在命令行提示符中输入“R”来运行R的控制台程序。

在Linux和UNIX平台中运行R

在Linux和UNIX中仅需要在命令行中输入“R”，便能打开R软件的控制台版本。键入时注意是R的大写，而不是小写字母r。

R软件包含大量命令选项，可以通过命令的后缀--help来详细查看。

另请参阅

方法1.4讲解如何退出R软件；有关当前工作目录，参见方法3.1；工作空间保存参见方法3.2；如何隐藏软件启动时给出的消息，请参见方法3.11。也可以参阅《R in a Nutshell》一书的第2章。

1.3 输入R命令

问题

R软件启动以后，得到命令提示符，该如何进行下一步的操作？

解决方案

可以简单地在命令提示符后输入表达式。R软件会自动计算并显示结果。也可以对已输入的内容进行命令行编辑，以方便输入。

讨论

R软件采用“>”符号提示输入指令代码，可以将R简单看做是一个能读取用户输入指令，运行计算并显示结果的“大计算器”。如：

```
> 1+1
[1]2
```

软件通过计算等式1+1，给出结果为2，并显示该结果。

2之前的[1]有些令人困扰。R软件始终以向量的形式输出结果，即使输出结果仅由一个元素组成。在上式的输出中，2之前的[1]，表示2是这一输出向量中的第一个元素。由于输出的向量仅由一个元素组成，因此这并不难理解。

在输入一个完整的表达式前，R软件会不断通过">"符号提示继续输入命令的余下部分。如max(1,3,5)为一个完整的表达式，因此R软件会根据输入的内容得到以下结果：

```
> max(1,3,5)
[1] 5
```

相反，"max(1,3,"不是一个完整的表达式，因此R软件会将">"符号改为"+"，提示继续输入上一行未完成的命令。如：

```
> max(1,3,
+5)
[1] 5
```

在输入的过程中很容易输错代码，而重新输入让人感到繁琐。R软件可以通过简易的命令编辑避免这样的情况，通过设定一些快捷键，完成对已输入语句的取消、修改和再次执行。我个人的命令行交互方式一般如下：

1. 输入R表达式（该表达式有错误）。
2. R报告有错误。
3. 通过键盘上的"↑"键调出之前输错的语句。
4. 通过"←"和"→"键移动光标，移动至输错的内容。
5. 通过Delete键删除输错的字符。
6. 在原语句中键入正确的字符。
7. 按下"回车"（Enter）键，再次执行该语句。

以上仅是最基本的操作。R软件提供了许多关于取消和编辑命令行的快捷方式，具体如表1-1所示。

表1-1：命令行编辑的快捷方式

键名	Ctrl键与其他键组合	效果
向上箭头	Ctrl+P	向前选择原有指令语句
向下箭头	Ctrl+N	向后选择原有指令语句
Backspace键	Ctrl+H	删除光标所在左侧的字符
删除键	Ctrl+D	删除光标所在右侧的字符

表1-1：命令行编辑的快捷方式（续）

键名	Ctrl键与其他键组合	效果
Home键	Ctrl+A	移动光标至指令开头
End键	Ctrl+E	移动光标至指令结尾处
向右箭头	Ctrl+F	光标向右移动一个字符
向左箭头	Ctrl+B	光标向左移动一个字符
	Ctrl+K	删除该命令光标所指处及之后的所有内容
	Ctrl+U	删除该命令光标所指处及之前的所有内容
Tab键		完成命名（限部分平台）

在Windows和OS X平台上，也可以使用鼠标选中命令，然后应用复制和粘贴命令把文本粘贴到新的命令行。

另请参阅

参见方法2.13。在Windows主菜单中，选择“帮助”（Help）→“控制台”（Console），可以得到完整的快捷键组合列表，对命令行编辑很有用。

1.4 退出R

问题

如何退出R。

解决方案

Windows

在菜单中选择“文件”（File）→“退出”（Exit），或点击窗口右上方红色的关闭程序按钮“×”。

OS X

按CMD+q（apple+q）组合键，或点击窗口左上方红色的关闭窗口按钮“×”。

Linux和UNIX

在命令行提示符下按Ctrl+D快捷键。

在所有操作系统中都可以使用q函数（*q*代表quit）来结束R程序。

```
> q()
```

注意，函数中必须包含一对空的圆括号。

讨论

退出时，R会提示你是否保存当前工作空间，你有三种选择：

- 保存工作空间并退出。
- 不保存，但退出。
- 取消退出，并返回到命令提示符。

选择保存后，R会在你的当前工作目录中生成一个后缀为*.RData*的文件。若当前工作空间中已存在同名文件，则会覆盖原有文件。因此在保存时需留心以免覆盖重要数据。

另请参阅

更多有关当前工作目录的细节请参见方法3.1；关于如何保存工作空间，请参见方法3.2；也可以参见《R in a Nutshell》一书中的第2章内容。

1.5 中断R正在运行的程序

问题

需要取消R软件正在进行的一个较长的计算，并回到命令提示符，但是不退出R。

解决方案

Windows或者OS X

可以选择按Esc键或点击“停止”（Stop-Sign）图标停止工作。

Linux和UNIX

按Ctrl+C键，中断R正在运行的程序而不退出R软件。

讨论

终止某一代码运行后，代码中的变量取值会根据程序运行的状态，保持到中断时计算的结果。可以在中断后通过工作空间进行查看。

另请参阅

参见方法1.4。

1.6 查看帮助文档

问题

如何查看R软件提供的帮助文档。

解决方案

通过输入`help.start`函数打开帮助文档的内容列表。即，

```
> help.start()
```

从中可以链接到所有已在本地安装的帮助文档。

讨论

R软件的基础发布版包含了数千个页面的帮助文件，并且当安装某一添加包时，其中的帮助文件也会自动安装到计算机中。

通过`help.start`函数可以轻松打开并查阅相关的帮助文件，该函数会打开一个置顶的帮助内容列表页面，如图1-2所示。

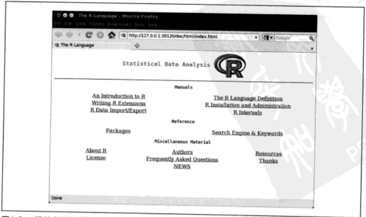


图1-2：帮助文档内容列表

该界面中“参考”（Reference）部分的两个链接较为常用：

R软件包（Packages）

点击查看已安装的所有R包列表，包含R基础包和之后安装的包。点击某一软件包名，可以查看该包的函数和数据集列表。

搜索引擎与关键字（Search Engine & Keywords）

点击进入简易搜索引擎页面，根据不同的关键字和短语搜索相应的帮助文档。该页面也给出了按照标题排序的常见关键词，点击它们会给出相关的帮助页面。

另请参阅

本地安装的帮助文件是安装R软件时R项目网站文档的复制品，网站上可能会有更新的帮助文件。

1.7 获取函数的帮助文档

问题

需要了解已经安装在计算机上的某个函数的更多信息。

解决方案

通过help函数查看该函数的帮助文档。例如，

```
> help(functionname)
```

用args函数快速获取函数的参数。例如，

```
> args(functionname)
```

用example函数查看函数的使用示例。例如，

```
> example(functionname)
```

讨论

本书将介绍许多R函数，但每个函数的全部功能之多，无法在本书中进行详细介绍。如果读者对某个函数感兴趣，我强烈推荐阅读关于该函数的帮助页面，你或许会从中获取有帮助的信息。

例如，想了解mean函数的全部功能，使用help命令查询该函数。例如：

```
> help(mean)
```

根据操作系统的不同，该命令可能打开函数的帮助文档，或者将函数的解释直接显示在R的控制台中。help命令的简写形式为在函数名前添加“?”符号，例如：

```
> ?mean
```

有时你仅想快速了解函数的参数：函数中有哪些参数？参数的顺序是什么？此时可使用args函数。例如：

```
> args(mean)
function (x, ...)
NULL
> args(sd)
function (x, na.rm = FALSE)
NULL
```

其中第一行的输出结果为该函数的调用简介。上面输出mean函数的调用简介，其中包含一个名为x的参数，它是一个数值向量。对于sd函数而言，args命令显示了它有和mean一样的一个x向量以及一个名为na.rm的可选参数（可以忽略第二行的结果，因为一般其结果都是NULL）。

R软件的函数帮助文档在最后一般都给出具体的使用例子。R的一个很诱人的特性是，可以使用example命令查看函数例子的具体执行，它可以演示该函数的功能。例如，对于mean函数的帮助文档，它含有一些示例，不必手动输入这些代码示例。只要使用example函数便能执行它，例如：

```
> example(mean)

mean> x <- c(0:10,50)

mean> xm <- mean(x)

mean> c(xm,mean(x,trim = 0.1))
[1] 8.75 5.50

mean> mean(USArrests,teim = 0.2)
    Murder  Assault  UrbanPop    rape
      7.42   167.60     66.20   22.16
```

以上过程中用户仅输入**example(mean)**命令，其余都由计算机完成。计算机会自动执行该函数在帮助文档中的示例代码，并给出相应的结果。

另请参阅

参考方法1.8来查找某一函数；参考方法3.5来查看更多搜索路径。

1.8 搜索帮助文档

问题

当在本地计算机的帮助文档中寻找已安装的某一函数的相关内容，但help函数显示没有找到该函数的任何搜索结果。

另外，你想用该函数的部分关键字搜索已安装的文档。

解决方案

使用help.search函数搜索本地计算机上安装的帮助文档：

```
> help.search("pattern")
```

此命令意在搜索名为*pattern*的某一函数或者关键字。注意：命令中*pattern*两边必须加上英文双引号。

上面的命令也可以使用两个问号的简易形式表达（这种情况下不需要添加双引号），例如：

```
> ??pattern
```

讨论

在搜索帮助文档的过程中，可能会遇到R软件无法给出任何搜索项的相关信息的情况，例如：

```
> help(adf.test)
No documentation for 'adf.test' in specified packages and libraries:
you could try 'help.search("adf.test")'
```

若你确定该函数安装在计算机中，则导致这样情况的原因可能是由于包含该函数的R包未载入，你不清楚哪个R包中包含这个函数。这有点类似于一个无法摆脱的困境（catch-22）（软件报错提示无法在搜索路径下找到该R包，因此R软件无法查询其帮助文件。详情参考方法3.5）。

要解决这样的问题，需要在所有已安装的R包中搜索该函数。使用错误消息中提示的方法来进行搜索（即用help.search命令），例如：

```
> help.search("adf.test")
```

搜索结果会给出所有包含该函数的R包列表。例如：

```
Help files with alias or concept or title matching 'adf.test' using
regular expression matching:

tseries::adf.test     Augmented Dickey-Fuller Test

Type '?PKG::FOO' to inspect entry 'PKG::FOO TITLE'.
```

以上结果显示了在tseries包中包含adf.test函数。可以再次使用help命令，在help的参数中明确指明包含需要帮助文档的函数所在的R包。例如：

```
> help(adf.test, package="tseries")
```

另外，也可以将tseries包添加至搜索列表中，并再次通过help命令进行最初的检索，它也会找到对应的函数并显示相关的帮助文件。

可以用关键字代替函数名来扩大搜索的范围。R软件会找出所有包含该关键字的帮助文档。比如你希望找出所有含有Augmented Dickey–Fuller (ADF)检验的函数，则可以通过以下命令：

```
> help.search("dickey-fuller")
```

由于我的计算机曾安装过两个包含ADF检验的数据包（fUnitRoots包和urca包），因此在我的计算机中上面的命令得到的结果为：

```
Help files with alias or conceot or title matching "dickey- fuller" using
fuzzy matching:

fUnitRoots::DickeyFullerPValues
                        Dickey-Fuller p Values
tseries::adf.test       Augmented Dickey-Fuller Test
urca::ur.df             Augmented-Dickey-Fuller Unit Root Test

Type '?PKG::FOO' to inspect entry 'PKG::FOO TITLE'.
```

另请参阅

可以通过文档浏览器来访问本地的搜索引擎，具体操作参见方法1.6。关于搜索路径请参考方法3.5。参考方法4.4来获得关于函数帮助的细节。

1.9 查看R软件包帮助信息

问题

如何了解已安装R软件包的相关信息。

解决方案

使用help命令并指定一个软件包名（而非指定一个函数名）。

```
> help(package="packagename")
```

讨论

有时你想知道某一软件包中包含的内容（函数和数据集），尤其是对于一个刚下载安装完的新R包时。因此，在得知R软件包名称后，可以使用help命令查看该软件包内容和其他信息。

使用以下help命令可以显示tseries软件包（R基本发布版中的标准软件包之一）的信息。

```
> help(package="tseries")
```

结果首先会给出数据包的简介，继而显示函数和数据集的索引。在我的计算机中，前几栏显示如下：

```
                Information on package 'tseries'

Description:

Package:       tseries
Version:       0.10-22
Date:                    2009-11-22
Title:                   Time series analysis and computational finance
Author:                  Compiled by Adrian Trapletti
                         <a.trapletti@swissonline.ch>
Maintainer:              Kurt Hornik <Kurt.Hornik@R-project.org>
Description:             Package for time series analysis and computational
                         finance
Depends:                 R (>= 2.4.0), quadprog, stats, zoo
Suggests:                its
Imports:                 graphics, stats, utils
License:                 GPL-2
Packaged:                2009-11-22 19:03:45 UTC; hornik
Repository:              CRAN
Date/Publication:        2009-11-22 19:06:50
Built:                   R 2.10.0; i386-pc-mingw32; 2009-12-01 19:32:47 UTC;
                         windows

Index:

NelPlo                   Nelson-Plosser Macroeconomic Time Series
USeconomic               U.S. Economic Variables
adf.test                 Augmented Dickey-Fuller Test
arma                     Fit ARMA Models to Time Series
```

```
.
. (etc.)
.
```

有些软件包中也会包含vignette——一些附加的文档，例如简介、教程或者开发文档等。这些vignette都会在安装软件包时，作为软件包的一部分安装到计算机中。vignette显示在软件包的帮助页面底部。

可以通过vignette命令查看计算机中包含的所有软件包的附加文档列表。

```
> vignette()
```

也可以通过在vignette命令中指定某一软件包的名称，查看特定软件包所附带的vignette。

```
> vignette(package="packagename")
```

每个vignette都对应一特定的名称，因此可以通过以下命令进行查看。

```
> vignette("vignettename")
```

另请参阅

对于如何通过帮助命令获得R软件包中某一函数的信息，参见方法1.7。

1.10 通过网络获取帮助

问题

如何通过网络搜索与R软件有关的信息和答案。

解决方案

在R软件中使用RSiteSearch函数对关键字或短语进行检索。

```
> RSiteSearch("key phrase")
```

在浏览器中，通过以下网站进行查询：

http://rseek.org

该网站是Google自定义搜索的，用于检索与R软件有关的内容。

http://stackoverflow.com/

Stack Overflow是一个针对编程有关的问题的Q&A网站，例如数据结构、代码以及图表绘制等编程问题的Q&A。

http://stats.stackexchange.com/

Stack Exchange也是一个可搜索的Q&A网站，但它更针对统计学的内容而非软件编程的问题。

讨论

使用`RSiteSearch`函数则会开启一个浏览器窗口，并连接至R项目网站（*http://search.r-project.org/*）的搜索引擎。从中你可以看到你的初始搜索，并可以完善它。例如，通过以下命令可以搜索"canonical correlation"：

```
> RSiteSearch("canonical correlation")
```

这种方法简单快捷并且不需借助R软件以外的工具，但它所能搜索到的结果仅限于R文档和邮件列表中的存档文件。

*rseek.org*网站提供了更广泛的搜索。其优点在于它借助了谷歌搜索引擎，并着重于与R相关的网站进行检索。它能删除许多普通谷歌检索到的无关结果。同时，*rseek.org*网站的美妙之处在于，它会将检索到的结果加以整理来显示。

图1-3显示了登录*rseek.org*后检索"canonical correlation"所得到的结果。页面左侧给出了一般的R网站的搜索结果，而页面右侧则将搜索到的结果分类整理为不同的标签分类：

- 简介（Introductions）
- 任务视图（Task Views）
- 支持列表（Support List）
- 函数（Functions）
- 书籍（Books）
- 博客（Blogs）
- 相关工具（Related Tools）

例如，可以点击"简介"（Introductions）标签来找到有关的教程材料。而"任务视图"（Task Views）一栏则会给出一些与关键字有关的任务视图资料。同样，点击"函数"（Functions）标签会看到有关的R函数链接。因此这一搜索方法能更好地帮助用户缩小检索范围。

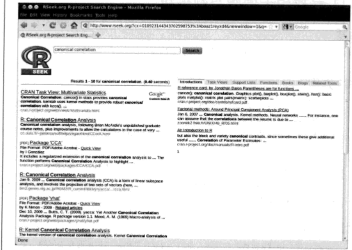


图1-3：搜索rseek.org的结果

Stack Overflow是一个Q&A网站，在其中所有用户都能进行提问，对其了解的人可以选择回答。通常每个问题都会有多个答案，浏览者可以投票选出最佳答案并将其置顶。这种机制使得Q&A网站具有大量的问答信息，并允许浏览者对此进行搜索。Stack Overflow是一个针对R软件编程问题所设立的Q&A网站。

Stack Overflow为许多编程语言存放问题。因此，在网站中进行搜索时应在搜索栏内加上“[r]”前缀指定对R软件的相关问题进行搜索。例如，如果搜寻关键字“[r] standard error”，它将在标签为R的有关问题中搜索，而排除那些与Python和C++编程语言相关的问题。

Stack Exchange网站（而非Stack Overflow网站）设立了统计分析的Q&A版面。其中的问题更侧重对统计方法的探讨，而编程语言则不是其探讨的重点。所以该网站的用户一般不会针对R软件提问，而是针对一些统计学的问题寻求帮助。

另请参阅

如果搜索结果给出的是一个有用的数据包，可以参考方法3.9安装该软件包。

1.11 寻找相关函数与数据包

问题

R有超过2000个软件包，你不知道如何选择合适的软件包。

解决方案

- 通过访问*http://cran.r-project.org/web/views/*网站中的任务视图，寻找并阅读与你研究相关的内容，其中能找到链接和相关软件包的简介。或者访问*http://rseek.org*网站，按照关键字搜索，点击“任务视图”（Task Views）标签来选择合适的任务视图。
- 访问crantastic网站（*http://crantastic.org*）并通过关键字检索合适的软件包。
- 访问*http://rseek.org*网站，检索名称或者关键字，并点击“函数”（Functions）标签，寻找相关的函数。

讨论

有一个问题往往会困扰R软件的初学者：你认为R软件能够解决你所遇到的问题，但不知道该选择哪种函数和软件包来执行它。在邮件询问列表中常常能见到这样的问题：“是否有R软件包能解决某问题？”会出现这种问题的原因在于，R软件提供过多的包，以至于让人不知所措。

在本书编写时就有超过2000个软件包能通过CRAN网站下载得到。每个软件包有一个概览页面，其中包括了对该软件包的简介以及软件包帮助文件的链接。当你对某个软件包感兴趣时，可以点击“参考手册”（Reference manual）的链接，查看PDF帮助文档寻找具体信息（概览页面同时也包含下载安装链接，但一般不采用这样的方法安装，具体参见方法3.9）。

有时我们仅对某一领域有兴趣，例如贝叶斯分析、计量经济学、运筹学或者绘图。CRAN网站中建有任务视图页面，它提供了软件包的简介。任务视图是一个能让你知道从何处开始的好地方，你从中可以了解哪些软件包可以应用。可以访问*http://cran.r-project.org/web/views/*网站查看任务视图；或者通过“解决方案”（Solution）中所描述的方法进行检索。

假设你已经知道某个有用的软件包的名字，则可以用上述方法进行查找。网站*http://cran.r-project.org/web/packages/*有所有R软件包按照字母序排列的列表，并有概览页面的链接。

另请参阅

可以下载安装一个名为sos的R软件包，它提供了强大的查询R包的其他方式；相关信息参考网站 *http://cran.r-project.org/web/packages/sos/vignettes/sos.pdf*。

1.12 查询邮件列表

问题

如何通过查询邮件列表中的档案来寻找问题的答案。

解决方案

- 打开*http://rseek.org*网站并输入与问题有关的关键字。检索结果出现后点击“支持列表”（Support Lists）标签。

  也可以在R软件中执行检索。通过RSiteSearch命令来进行检索：

  ```
  >RSiteSearch("keyphrase")
  ```

- 系统会打开一个新的窗口显示检索结果。在“Target”下方勾选R语言的帮助信息源，并取消其他信息源的勾选，随后重新提交查询。

讨论

这个方法只是方法1.10的一个应用。由于你希望解决的问题很可能在邮件列表中已存在答案，因此对邮件列表的检索过程很重要，这样只需检索而不用添加新的问题。

另请参阅

CRAN网站还提供了一系列其他有关搜索网站的资源，详情请访问网址：*http://cran.r-project.org/search.html*。

1.13 向邮件列表提交问题

问题

如何通过R帮助邮件列表向R社区提交问题来寻求帮助。

解决方案

“邮件列表”（Mailing Lists）页面有一个关于使用“R帮助”（R-help）列表的基本信息以及使用该邮件列表的指南。以下是基本的操作流程：

1. 在Main R Mailing List中订阅“R帮助”（R-help）列表。
2. 阅读关于如何书写有效问题的“发帖指南”（Posting Guide）。
3. 仔细并正确地书写问题。如果合适，包含一个最小的自我包含的例子，这样其他人可以重现你遇到的错误或问题。
4. 将你的问题发送至*r-help@r-project.org*。

讨论

R邮件列表是一个强大的资源。但是最好将这种方法视为最后的一条帮助途径。在这之前先阅读帮助页面和帮助文档，或者搜索列表存档文件，搜索网络。很有可能你所要找的问题先前已经有人回答，很少能遇到自己想到而别人想不到的问题。

写好问题之后，提交问题则很简单。只要将你的问题发送至*r-help@r-project.org*即可。注意你必须订阅邮件才能有资格进行提问，否则你所提交的问卷可能不会被接受。

如果你的问题是由于R软件代码在执行时出错或者非期望的结果出现，这时问题中要包含一个最小的自我包含的例子：

最少

　　使用最少的代码片段来展示你的问题，并删除一切与问题不相关的内容。

自我包含

　　包含必要的数据以确切地再现问题中的错误。若列表的读者在阅读过程中无法重现问题，则无法对问题进行诊断。对于那些复杂的数据结构，用`dump`命令将数据另存为一个ASCII文档，并将其包含至你的列表求助消息中。

在问题中添加示例，能清楚地说明你所问的问题，提高获得有效帮助的机会。

事实上，除了“R帮助”（R-help）这个主要列表外，还有其他多种邮件列表。同时，也有许多针对某一特定领域的特别兴趣小组，如遗传学、金融学、R软件开发，甚至与R软件相关的工作。可通过*https://stat.ethz.ch/mailman/listinfo*网址查询完整的列表。如果你的问题针对某一特定领域，那么通过查阅和选择这些领域的相关列表，可能会得到一个更好的答案。但对于“R帮助”（R-help）来说，建议事先查阅其他的邮件列表存档文件，若检索不到相关的答案时再选择提交自己的问题。

另请参阅

推荐读者在提交自己的问题前，先阅读一篇由Eric Raymond和Rick Moen写的一篇名为“How to Ask Questions the Smart Way”的优秀文章。

第2章

基础知识

简介

本章中所介绍的方法，既包含了解决问题的思想，也包含了解决问题的一些基本技巧。诚然，本章中所举的案例都只是一些最常见的问题，但通过介绍这些问题我们能了解R软件编程的基本知识和语法。这些知识适用于包括本书R代码在内的几乎所有代码。因此建议R软件初学者简要浏览本章，对一些语法规定有所了解。

2.1 显示内容

问题

如何通过R软件显示某一变量或表达式的值。

解决方案

在提示符后直接输入变量名或表达式，R软件便会直接在屏幕中输出其值。使用print函数能输出所有的变量和表达式值，使用cat函数则能选择以用户自定义的格式显示对象的值。

讨论

通过R软件显示结果的步骤很简单：只需在提示符后输入变量名或表达式：

```
> pi
[1] 3.141593
```

```
> sqrt(2)
[1] 1.414214
```

当输入如上表达式后，R软件会对表达式进行计算并自动调用print函数显示结果。因此上述命令等同于如下命令：

```
> print(pi)
[1] 3.141593
> print(sqrt(2))
[1] 1.414214
```

print函数的优点在于它知道该以何种格式显示结果，包括一些具有格式的变量，如矩阵和列表：

```
> print(matrix(c(1,2,3,4), 2, 2))
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> print(list("a","b","c"))
[[1]]
[1] "a"

[[2]]
[1] "b"

[[3]]
[1] "c"
```

这一命令很常用，因为你可以随时通过print函数显示数据，而不必在意数据显示的顺序与逻辑，即使对如矩阵这样复杂的数据格式也是如此。

但print函数也有其局限性：print函数每次只能显示一个对象。同时显示多个变量会得到如下结果：

```
> print("The zero occurs at", 2*pi, "radians.")
Error in print.default("The zero occurs at", 2 * pi, "radians.") :
        unimplemented type 'character' in 'asLogical'
```

只有通过多次使用print函数才能显示多个对象，但用户往往会认为这种方法过于繁琐：

```
> print("The zero occurs at"); print(2*pi); print("radians")
[1] "The zero occurs at"
[1] 6.283185
[1] "radians"
```

cat函数是另外一种可以替代print的显示方式。它可以将多个对象连接并以连续的方式显示：

```
> cat("The zero occurs at", 2*pi, "radians.", "\n")
The zero occurs at 6.283185 radians.
```

注意，cat函数默认在两个对象间加上空格。如果需要换行，则可以使用换行符（\n）来结束本行语句。

cat函数也能显示简单向量：

```
> fib <- c(0,1,1,2,3,5,8,13,21,34)
> cat("The first few Fibonacci numbers are:", fib, "...\n")
The first few Fibonacci numbers are: 0 1 1 2 3 5 8 13 21 34 ...
```

cat函数对变量的输出有更多的控制和选择，这在R程序中尤为重要。但它也有缺陷，即cat函数无法显示复合的数据结构，如矩阵和列表。用cat函数显示列表会得到以下结果：

```
> cat(list("a","b","c"))
Error in cat(list(...), file, sep, fill, labels, append) :
        argument 1 (type list) cannot be handled by cat
```

另请参阅

关于如何控制输出变量的格式，请参见方法4.2。

2.2 设定变量

问题

如何将某个值赋值给一个变量。

解决方案

使用赋值运算符（<-）进行赋值。在赋值前无须对变量进行声明：

```
> x <- 3
```

讨论

R软件采用“计算器”模式，方便快捷。但是，有时候需要定义变量并保存变量值。这省去了重复输入的时间并使你的工作更为明晰。

在R软件中，不必对变量进行声明或者显式地创建变量，只需要将值赋予一个名称，R软件就会自动生成该名称的变量：

```
> x <- 3
> y <- 4
> z <- sqrt(x^2 + y^2)
> print(z)
[1] 5
```

注意，赋值操作由一个小于号（<）和连字符（-）构成，两个符号之间没有空格。

当使用此种方法定义变量时，该变量将存储到当前的工作空间中。此时工作空间仅存储在计算机的内存中，当退出R软件时可保存至本地硬盘。工作空间会永久保存该变量，直至用户删除或替代该变量。

R软件是动态的输入语言，即可随意改变变量的数据类型。我们可以先定义x为数值型变量，随后马上对其赋值一个字符串向量，在这一过程中R软件能完全理解用户的意图：

```
> x <- 3
> print(x)
[1] 3
> x <- c("fee", "fie", "foe", "fum")
> print(x)
[1] "fee" "fie" "foe" "fum"
```

在某些R函数中，你会看到很特别的赋值符号<<-：

```
x <<- 3
```

这一操作能强制赋值给一个全局变量，而不是局部变量。

为了全面介绍，在此给出另外两种赋值形式。也可以在命令提示符中使用单个等号（=）对变量进行赋值。在所有可以应用向左赋值符号（<-）的地方都可以使用向右赋值符号（->），它对右侧变量进行赋值：

```
> foo = 3
> print(foo)
[1] 3
> 5 -> fum
> print(fum)
[1] 5
```

但上述两种赋值方法不会出现在本书中，也建议读者尽量不使用。等号赋值很容易与假设检验中的等号混淆。而右侧赋值并不常见，并且会使较长的编程语句难以阅读和理解。

另请参阅

关于assign函数的帮助页面，请参考方法2.4、方法2.14和方法3.2。

2.3 列出所有变量

问题

你希望知道目前工作空间中存在哪些已定义的变量和函数。

解决方案

使用ls函数，或者使用ls.str函数了解每个变量更详细的信息。

讨论

ls函数可以显示当前工作空间中所有对象的名称：

```
> x <- 10
> y <- 50
> z <- c("three", "blind", "mice")
> f <- function(n,p) sqrt(p*(1-p)/n)
> ls()
[1] "f" "x" "y" "z"
```

注意，ls函数输出的结果是一个字符串向量，其中向量的每个元素代表一个变量名。当工作空间中没有已定义的变量时，函数ls会返回一个空向量，它会产生如下令人迷惑的结果：

```
> ls()
character(0)
```

事实上，R软件采用这样的方式向用户说明，**ls**函数返回一个长度为0的字符串向量，即工作空间中不含有任何已定义变量。

如果你除了变量名称以外还想对变量有更多的了解，那么你可以使用ls.str函数，该函数会返回变量的一些其他信息：

```
> ls.str()
f : function (n, p)
x :  num 10
y :  num 50
z :  chr [1:3] "three" "blind" "mice"
```

ls.str函数之所以写为ls.str，原因在于其功能既显示了所有变量的名称，又对所有变量使用了str函数，方法12.15对此进行了详细的说明。

ls函数不会显示以点（.）开头的变量名，以点开头的变量一般作为隐藏变量不为用户

所知（这一输出规定来源于UNIX系统）。在R软件中，可以通过将ls.str函数中的all.names参数设定为TRUE，强制列出所有变量：

```
> .hidvar <- 10
> ls()
[1] "f" "x" "y" "z"
> ls(all.names=TRUE)
[1] ".hidvar" "f"       "x"       "y"       "z"
```

另请参阅

方法2.4介绍了如何删除变量，方法12.15介绍了如何检查某一变量。

2.4 删除变量

问题

你希望删除工作空间中不需要的变量和函数，或者完全删除它们的取值内容。

解决方案

使用rm函数。

讨论

在R软件的使用过程中，工作空间容易很快变得杂乱。rm函数能永久地从工作空间中删除一个或多个对象：

```
> x <- 2*pi
> x
[1] 6.283185
> rm(x)
> x
Error: object "x" not found
```

该命令无法“撤销”，即删除的变量无法找回。

你可以通过如下命令同时删除多个变量：

```
> rm(x,y,z)
```

你甚至可以同时删除工作空间中所有的内容。rm函数中有一个list参数，它包含所有需要删除的变量名称。前面章节介绍过ls函数能返回所有变量名称，因此你可以通过结合rm函数与ls函数，删除工作空间中的所有变量：

```
> ls()
[1] "f" "x" "y" "z"
> rm(list=ls())
> ls()
character(0)
```

> **R软件的道德规范**
>
> 在与他人共享代码时，绝不能将带有rm(list=ls())的恶意代码通过网络示例或者其他方法发送至他人。用此手段删除对方工作空间中所有的内容是如此粗鄙的行为，让你变得不受欢迎。

另请参阅

参考方法2.3。

2.5 生成向量

问题

如何生成一个向量。

解决方案

通过c(...)命令对给定的值构建一个向量。

讨论

向量不仅是R的一种数据结构，它还是贯通R软件的重要组成部分。向量中可以包含数值、字符串或者逻辑值，但不能由多种格式混合组成。

在c(...)命令中添加元素对向量进行赋值：

```
> c(1,1,2,3,5,8,13,21)
[1]  1  1  2  3  5  8 13 21
> c(1*pi, 2*pi, 3*pi, 4*pi)
[1]  3.141593  6.283185  9.424778 12.566371
> c("Everyone", "loves", "stats.")
[1] "Everyone" "loves"    "stats."
> c(TRUE,TRUE,FALSE,TRUE)
[1]  TRUE  TRUE FALSE  TRUE
```

如果c(...) 中的参数自身是向量，那么c(...) 命令会将多个向量合为一个向量：

```
> v1 <- c(1,2,3)
> v2 <- c(4,5,6)
> c(v1,v2)
[1] 1 2 3 4 5 6
```

对于一个向量来说，其中的内容不能由多种数据格式混合组成，如在一个向量中同时包含数值和字符串。R软件对于混合型向量会进行如下的格式转换：

```
> v1 <- c(1,2,3)
> v3 <- c("A","B","C")
> c(v1,v3)
[1] "1" "2" "3" "A" "B" "C"
```

这里，用户希望将一组数值数据和一组字符串数据同时赋值给一个新的向量。对于这种情况，R软件会先将数值数据转换为字符串数据，使得两组数据的类型得以统一。

理论上来说，两组数据能同时赋值于一个向量的条件，在于两组数据具有相同的类型（mode）。例如3.1415和"foo"分别为数值型和字符型：

```
> mode(3.1415)
[1] "numeric"
> mode("foo")
[1] "character"
```

上述两者的类型不同。为了生成新的向量，R软件将3.1415转换为字符类型，使得3.1415的类型与"foo"的类型一样：

```
> c(3.1415, "foo")
[1] "3.1415" "foo"
> mode(c(3.1415, "foo"))
[1] "character"
```

警告： c是一个通用的运算符，这意味着它不仅应用于向量，同时也应用于其他的数据类型。但是，它可能不是那么精确地与用户预期相一致。因此在将c命令用于其他数据类型和对象前，要查看它的效果。

另请参阅

更多有关向量和其他数据类型的内容，请查看第5章的简介部分。

2.6 计算基本统计量

问题

如何使用R软件计算下列统计量：均值、中位数、标准差、方差、协方差和相关系数。

解决方案

采用如下函数进行计算，其中x、y均为向量：

- `mean(x)`
- `median(x)`
- `sd(x)`
- `var(x)`
- `cor(x, y)`
- `cov(x, y)`

讨论

我初次阅读R软件帮助文件是为了寻找“标准差的计算过程”这一内容，原本认为帮助文件会以一整章篇幅介绍这一重要概念。

实际上没有那么复杂。

R软件中，用简单的函数便能完成标准差和其他基本统计量的计算。一般来说，函数参数是一个数值向量，而函数返回计算出的统计量：

```
> x <- c(0,1,1,2,3,5,8,13,21,34)
> mean(x)
[1] 8.8
> median(x)
[1] 4
> sd(x)
[1] 11.03328
> var(x)
[1] 121.7333
```

其中sd函数计算样本标准差，var函数计算样本方差。

cor函数以及cov函数分别计算两变量间的相关系数与协方差：

```
> x <- c(0,1,1,2,3,5,8,13,21,34)
> y <- log(x+1)
> cor(x,y)
[1] 0.9068053
> cov(x,y)
[1] 11.49988
```

上述函数对于是否存在缺失值（NA）很敏感。某个变量中的一个缺失值就有可能导致函数返回NA结果，甚至可能造成计算机在计算过程中报错：

```
> x <- c(0,1,1,2,3,NA)
> mean(x)
[1] NA
> sd(x)
[1] NA
```

虽然R软件对于缺失值的敏感程度有时会造成用户的不便，但这种处理方式也是合情合理的。对于R软件返回的结果你应该慎重地考虑：数据中的缺失值是否会严重影响统计结果？如果是，那么R软件返回错误结果是正确的；如果不是，则可以通过设置参数na.rm=TRUE，告知R软件忽略缺失值：

```
> x <- c(0,1,1,2,3,NA)
> mean(x, na.rm=TRUE)
[1] 1.4
> sd(x, na.rm=TRUE)
[1] 1.140175
```

mean函数和sd函数能巧妙地处理数据框数据，自动将数据框中的每一列认为是不同的变量，并对每列数据分别进行计算。下面的例子展示了mean和sd函数对有三列的数据框的计算结果：

```
> print(dframe)
       small    medium       big
1  0.6739635 10.526448  99.83624
2  1.5524619  9.205156 100.70852
3  0.3250562 11.427756  99.73202
4  1.2143595  8.533180  98.53608
5  1.3107692  9.763317 100.74444
6  2.1739663  9.806662  98.58961
7  1.6187899  9.150245 100.46707
8  0.8872657 10.058465  99.88068
9  1.9170283  9.182330 100.46724
10 0.7767406  7.949692 100.49814
> mean(dframe)
    small    medium       big
 1.245040  9.560325 99.946003
> sd(dframe)
    small    medium       big
0.5844025 0.9920281 0.8135498
```

注意，mean和sd函数都会返回3个值，每个数值对应着对数据框中一列数据的计算结果（一般地，R软件会以一个包含三个元素的向量返回结果，其中每个元素的names属性由数据框中各个列的名称得来）。

var函数也能处理数据框数据，但处理方式与mean函数和sd函数有些许不同。var函数计算每两列变量间的协方差，并以协方差矩阵的形式返回结果：

```
> var(dframe)
```

```
             small      medium         big
small   0.34152627 -0.21516416 -0.04005275
medium -0.21516416  0.98411974 -0.09253855
big    -0.04005275 -0.09253855  0.66186326
```

同样，如果x是一个数据框或矩阵，则cor(x) 返回其相关系数矩阵；而cov(x) 返回其协方差矩阵：

```
> cor(dframe)
             small      medium         big
small   1.00000000 -0.3711367 -0.08424345
medium -0.37113670  1.0000000 -0.11466070
big    -0.08424345 -0.1146607  1.00000000
> cov(dframe)
             small      medium         big
small   0.34152627 -0.21516416 -0.04005275
medium -0.21516416  0.98411974 -0.09253855
big    -0.04005275 -0.09253855  0.66186326
```

median函数无法辨认数据框形式的数据。若需计算数据框数据的中位数，需要使用方法6.4对各列分别进行计算。

另请参阅

参见方法2.14、方法6.4和方法9.17。

2.7 生成数列

问题

如何生成一个数列。

解决方案

使用表达式n:m生成简单数列*n*,*n*+1,*n*+2,...,*m*：

```
> 1:5
[1] 1 2 3 4 5
```

对于增量不为1的数列，可以使用seq函数：

```
> seq(from=1, to=5, by=2)
[1] 1 3 5
```

使用rep函数生成由一个数的重复所组成的数列：

```
> rep(1, times=5)
[1] 1 1 1 1 1
```

讨论

冒号运算符（n:m）会生成包含n,n+1,n+2,...,m的一个向量：

```
> 0:9
[1] 0 1 2 3 4 5 6 7 8 9
> 10:19
[1] 10 11 12 13 14 15 16 17 18 19
> 9:0
[1] 9 8 7 6 5 4 3 2 1 0
```

注意，上述最后一个表达式（9:0），R软件能自动识别9大于0并以递减的形式生成数列。

冒号运算符仅能生成增量为1的数列。而seq函数通过它的第三个参数来规定数列元素的增量：

```
> seq(from=0, to=20)
[1]  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
> seq(from=0, to=20, by=2)
[1]  0  2  4  6  8 10 12 14 16 18 20
> seq(from=0, to=20, by=5)
[1]  0  5 10 15 20
```

相应地，你可以在函数中规定输出数列的长度，R软件会自动识别并根据要求生成等增量数列：

```
> seq(from=0, to=20, length.out=5)
[1]  0  5 10 15 20
> seq(from=0, to=100, length.out=5)
[1]   0  25  50  75 100
```

函数seq的增量参数并非一定是整数。R软件也可以生成具有分数增量的数列：

```
> seq(from=1.0, to=2.0, length.out=5)
[1] 1.00 1.25 1.50 1.75 2.00
```

特殊情况下，若需要生成重复某个值的数列，则可以使用rep函数，生成的数列重复其第一个参数值：

```
> rep(pi, times=5)
[1] 3.141593 3.141593 3.141593 3.141593 3.141593
```

另请参阅

若需要生成日期型格式的数列，请参见方法7.14。

2.8 向量比较

问题

如何比较两个向量，或者将一个向量的所有元素与某一个常数进行比较。

解决方案

比较运算符（==、!=、<、>、<=、>=）能对两向量间的各个元素进行比较。这些运算符也能将向量中所有元素与一个常数进行比较。返回结果是每两个元素间比较结果的逻辑值向量。

讨论

R软件包含两个逻辑值，TRUE和FALSE。在其他编程语言中也称为布尔值（Boolean values）。

比较运算符通过比较两个值，并根据比较结果返回TRUE或FALSE：

```
> a <- 3
> a == pi       # 检验两者是否相等
[1] FALSE
> a != pi       # 检验两者是否不等
[1] TRUE
> a < pi
[1] TRUE
> a > pi
[1] FALSE
> a <= pi
[1] TRUE
> a >= pi
[1] FALSE
```

你可以使用R软件一次性地对两个向量进行比较，它会将两个向量中每两个对应的元素进行比较，并以逻辑值向量方式返回比较结果：

```
> v <- c( 3, pi,  4)
> w <- c(pi, pi, pi)
> v == w                       # 比较两个各自包含3个元素的向量
[1] FALSE  TRUE FALSE          # 结果以包含3个逻辑值的向量形式输出
> v != w
[1]  TRUE FALSE  TRUE
```

```
> v < w
[1]  TRUE FALSE FALSE
> v <= w
[1]  TRUE  TRUE FALSE
> v > w
[1] FALSE FALSE  TRUE
> v >= w
[1] FALSE  TRUE  TRUE
```

也可以将一个向量与一个常数进行比较，R软件会将常数扩充为一组长度与所比较向量的长度相等，并由常数值重复组成的向量，再将新向量与它需要比较向量的对应元素进行比较。所以，之前的例子可以简化为：

```
> v <- c(3, pi, 4)
> v == pi                        # 将包含3个元素的向量与一个常数进行比较
[1] FALSE  TRUE FALSE
> v != pi
[1]  TRUE FALSE  TRUE
```

（这里是循环规则的应用，参见方法5.3。）

比较两个向量后，你通常会想知道比较结果中是否存在TRUE，或者比较结果是否全为TRUE。可以应用函数any和all来检验上述问题。两个函数都针对逻辑型变量进行检验，其中如果元素中含有至少一个TRUE，则any函数返回TRUE；如果元素全为TRUE，则all函数返回TRUE：

```
> v <- c(3, pi, 4)
> any(v == pi)                   # 若v向量中元素至少一个等于pi，则返回TRUE
[1] TRUE
> all(v == 0)                    # 若v向量中所有元素都为0，则返回TRUE
[1] FALSE
```

另请参阅

参见方法2.9。

2.9 选取向量中的元素

问题

如何选取向量中一个或多个元素。

解决方案

选择适合问题的索引技术：

- 根据元素在向量中的位置使用方括号来选出元素，如v[3]代表了v向量中的第三个元素。
- 索引前加负号（-），排除向量中相应位置的元素。
- 使用向量索引来选择多个元素值。
- 使用逻辑向量根据条件来选择元素。
- 使用名称来选择命名的元素。

讨论

从向量中选出某些元素是R的又一项强大功能。和其他编程语言一样，R选取向量元素的基本方法是使用一对方括号和简单索引（下标）：

```
> fib <- c(0,1,1,2,3,5,8,13,21,34)
> fib
[1]  0  1  1  2  3  5  8 13 21 34
> fib[1]
[1] 0
> fib[2]
[1] 1
> fib[3]
[1] 1
> fib[4]
[1] 2
> fib[5]
[1] 3
```

注意，向量第一个元素的索引（或下标）为1，而非某些编程语言中的0。

另外也可以同时选择一个向量中的多个元素，向量的索引本身可以是一个向量，并且根据下标向量中所指定的位置选择原向量中的元素：

```
> fib[1:3]                          # 选择下标为1至3的元素
[1] 0 1 1
> fib[4:9]                          # 选择下标为4至9的元素
[1]  2  3  5  8 13 21
```

1:3这样的下标意味着选择第1、2、3个元素，如上例所示。索引向量可以不是简单数列，可以选择向量数据中的任何元素，如下例所示，它选择第1、2、4和第8个元素：

```
> fib[c(1,2,4,8)]
[1]  0  1  2 13
```

R将负索引看做是排除向量中相应索引的元素。如下标为-1，意味着选择除了向量的第一个元素外的所有其他元素：

```
> fib[-1]          # 忽略第一个元素
[1]  1  1  2  3  5  8 13 21 34
```

通过使用负索引的索引向量，该方法可以扩展为排除多个元素：

```
> fib[1:3]         # 前述向量
[1] 0 1 1
> fib[-(1:3)]      # 在索引前添加负号，排除相应的元素
[1]  2  3  5  8 13 21 34
```

还可以使用逻辑向量从数据向量中选择元素。与索引逻辑向量取值为TRUE的元素相对应的原始数据向量的元素将被选择：

```
> fib < 10                        # 仅当向量中元素值小于10时该表达式为TRUE
[1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
> fib[fib < 10]                   # 使用该表达式选择向量中小于10的值
[1] 0 1 1 2 3 5 8

> fib %% 2 == 0                   # 仅当向量fib中元素值为偶数时该表达式为TRUE
[1]  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE  TRUE
> fib[fib %% 2 == 0]              # 使用该表达式选择向量中的偶数值
[1]  0  2  8  34
```

一般地，索引逻辑向量的长度应与原始数据向量的长度相同，这样才能清晰地选择或者排除每一个元素（若两者长度不同，则需要了解循环规则，具体请参见方法5.3）。

结合向量比较、逻辑运算符以及向量索引，可以用少量的R命令来完成强大的选择功能：

选择所有小于中位数的元素

```
v[ v > median(v) ]
```

选择分布于两端5%的元素

```
v[ (v < quantile(v,0.05)) | (v > quantile(v,0.95)) ]
```

选择所有处于均值的两倍标准差区间以外的元素

```
v[ abs(v-mean(v)) > 2*sd(v) ]
```

选择所有NA或NULL值的元素

```
v[ !is.na(v) & !is.null(v) ]
```

最后一个索引特征可以让你通过名称选择元素。它要求数据向量有name属性，即其中每个元素都定义各自的名称。可以指定一个字符串向量来完成该数据向量名称的定义：

```
> years <- c(1960, 1964, 1976, 1994)
> names(years) <- c("Kennedy", "Johnson", "Carter", "Clinton")
> years
Kennedy Johnson  Carter Clinton
   1960    1964    1976    1994
```

一旦元素的名称已定义，则可以使用名称引用向量中的元素：

```
> years["Carter"]
Carter
  1976
> years["Clinton"]
Clinton
    1994
```

可以推广到使用名称来索引向量元素：R软件会返回向量中对应名称的元素：

```
> years[c("Carter","Clinton")]
Carter Clinton
  1976    1994
```

另请参阅

有关循环规则的细节，请参阅方法5.3。

2.10 向量的计算

问题

你希望对整个向量执行计算。

解决方案

基本的数学运算符可以对向量中的元素进行逐个计算。许多其他的函数也能对向量元素逐个进行运算，并以向量的形式输出结果。

讨论

向量计算是R软件的一大特色。所有的基本数学运算符都能应用于向量对中。这些运算符对两个向量中相应的每个元素对进行计算，即将两个向量中对应的元素进行基本运算：

```
> v <- c(11,12,13,14,15)
> w <- c(1,2,3,4,5)
> v + w
[1] 12 14 16 18 20
> v - w
[1] 10 10 10 10 10
> v * w
[1] 11 24 39 56 75
> v / w
[1] 11.000000  6.000000  4.333333  3.500000  3.000000
```

```
> w ^ v
[1]      1    4096    1594323    268435456  30517578125
```

注意，输出的结果向量的长度与原向量的长度相等。原因是结果向量中的每个元素都是由原向量对中对应的两个元素计算得来。

若使一个向量与一个常数进行运算，则会将该向量的每个元素与常数进行运算：

```
> w
[1] 1 2 3 4 5
> w + 2
[1] 3 4 5 6 7
> w - 2
[1] -1  0  1  2  3
> w * 2
[1]  2  4  6  8 10
> w / 2
[1] 0.5 1.0 1.5 2.0 2.5
> w ^ 2
[1]  1  4  9 16 25
> 2 ^ w
[1]  2  4  8 16 32
```

例如，可以在一个表达式中得到一个向量减去其元素均值后的向量：

```
> w
[1] 1 2 3 4 5
> mean(w)
[1] 3
> w - mean(w)
[1] -2 -1  0  1  2
```

同样，可以通过计算向量减去其均值并除以其标准差，来获得该向量数据的z分数（z-score）：

```
> w
[1] 1 2 3 4 5
> sd(w)
[1] 1.581139
> (w - mean(w)) / sd(w)
[1] -1.2649111 -0.6324555  0.0000000  0.6324555  1.2649111
```

向量的运算功能远不止对元素的简单运算。还有许多函数对整个向量进行运算。如sqrt函数和log函数，都可以应用于整个向量中的每个元素，并以向量的形式输出结果：

```
> w
[1] 1 2 3 4 5
> sqrt(w)
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
> log(w)
[1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379
```

```
> sin(w)
[1]  0.8414710  0.9092974  0.1411200 -0.7568025 -0.9589243
```

R软件的向量运算有两大优点。第一个最明显的优点是操作的简便性，其他编程软件中需要通过循环才能完成的操作，在R软件中一行命令便可以实现。第二个优点是计算速度快。大多数向量化的运算直接由C语言代码来实现，它比你自己用R写的代码本质上快很多。

另请参阅

向量与常量之间的运算是循环规则的一个特例，请参见方法5.3。

2.11 运算符优先级问题

问题

R软件输出结果有误，你希望了解问题是否由运算符的优先级所导致的。

解决方案

所有的运算符显示在表2-1中，并以最高优先级至最低优先级的顺序排列。相同优先级的运算符，除特指外皆由从左至右的顺序进行运算。

表2-1：运算符优先级

运算符	含义	参考
[[[	索引	方法2.9
:: :::	使用名称访问变量	
$ @	元素提取、位置提取	
^	指数形式（从右到左）	
- +	元素的负、正	
:	创建数列	方法2.7，7.14
%any%	特殊运算符	讨论
* /	乘、除	讨论
+ -	加、减	
== != < > <= >=	比较运算符	方法2.8
!	逻辑取反	
& &&	逻辑“与”、短路“与”	

表2-1：运算符优先级（续）

运算符	含义	参考
| ||	逻辑"或"、短路"或"	
~	公式	方法11.1
-> ->>	向右赋值	方法2.2
=	赋值（从右向左）	方法2.2
<- <<-	赋值（从右向左）	方法2.2
?	帮助	方法1.7

讨论

用户在R中搞错运算符的优先级是经常遇到的问题。我经常会犯这样的错误，例如我会不假思索地认为表达式`0:n−1`会生成从0~*n*−1的数列，但事实并非如此：

```
> n <-10
> 0:n-1
[1] -1  0  1  2  3  4  5  6  7  8  9
```

该表达式生成−1~*n*−1的数列，因为R软件将上式理解为`(0:n)−1`。

你可能不熟悉表2-1中的符号`%any%`，R中用两个百分号夹带一个符号的形式（`%...%`）表示一个二元运算符。R中预定义的二元运算符的含义如下：

`%%`

取模

`%/%`

整除

`%*%`

矩阵乘积

`%in%`

右侧变量中包含左侧变量时，为`TRUE`；否则，为`FALSE`。

你可以通过`%...%`记号来定义新的二元运算符，参见方法12.19。此种运算符都具有相同的运算优先级。

另请参阅

关于向量之间运算的细节，请参阅方法2.10；矩阵的运算，参阅方法5.15；关于如何定

义所需要的运算符，参见方法12.19。可以查看R软件中算术运算与语法的帮助页面，也可以参阅《R in a Nutshell》（O'Reilly）一书第5章和第6章的内容。

2.12 定义函数

问题

如何定义一个R函数。

解决方案

使用关键字function，并在其后跟随函数参数列表和函数主体。其基本形式如下：

```
function(param1, ...., paramN) expr
```

函数主体可以是一系列表达式，这些表达式需要用大括号括起来：

```
function(param1, ..., paramN) {
    expr1
    .
    .
    .
    exprM
}
```

讨论

函数的定义告诉R软件“用何种方式进行计算”。例如，R软件没有内置计算变异系数的函数，因此你可以定义函数如下：

```
> cv <- function(x) sd(x)/mean(x)
> cv(1:10)
[1] 0.5504819
```

第一行定义了名为cv的函数，第二行引用该函数，以1:10作为其参数x的值。函数对参数应用函数主体中的表达式sd(x)/mean(x)进行计算并返回结果。

定义函数后，我们可以在任何需要函数的地方应用它，例如可以作为lapply函数的第二个参数（参见方法6.2）：

```
> cv <- function(x) sd(x)/mean(x)
> lapply(lst, cv)
```

函数主体如果包含多行表达式，则需要使用大括号来确定函数内容的起始和结束位置。下面这一函数采用了欧几里德算法计算两个整数的最大公约数：

```
> gcd <- function(a,b) {
+   if (b == 0) return(a)
+   else return(gcd(b, a %% b))
+ }
```

R软件也允许使用匿名函数，匿名函数是没有函数名称但在单行的语句中很实用的函数。先前的例子中我们提到将cv函数作为lapply函数的一个参数，而若使用匿名函数直接作为lapply函数的参数，则能将原先的命令简化至同一行中：

```
> lapply(lst, function(x) sd(x)/mean(x))
```

由于本书重点不在于介绍R的编程语言，这里不对R函数编程的细微之处进行解释。下面给出几个需要注意的地方：

返回值

所有函数都有一个返回值，即函数主体最后一个表达式值。你也可以通过return(*expr*)命令给出函数的返回值。

值调用

函数参数是“值调用”——如果你改变了函数中的参数值，改变只是局部的，并不会影响该参数所引用的变量值。

局部变量

你可以简单地通过赋值来创建一个局部变量，函数结束后该局部变量会消失。

条件执行

R语法中包含if语句，更多详情可以使用help(Control)命令查看。

循环语句

R语法中也包括for循环、while循环以及repeat循环语句。更多详情可以使用help(Control)命令查看。

全局变量

在函数中，你可以通过<<-操作符来改变全局变量的值，但此种方法不推荐使用。

另请参阅

有关如何定义函数，参见《An Introduction to R》（*http://cran.r-project.org/doc/manuals/R-intro.pdf*）和《R in a Nutshell》。

2.13 减少输入，得到更多命令

问题

你对于输入大量长长的命令，尤其是不断重复输入同一命令感到枯燥。

解决方案

打开一个编辑窗口将经常使用的R代码存放于其中，并直接从该窗口中执行这些命令。在命令行仅仅输入一些简短的或者一次性的命令。

完成分析后，你可以把积累的命令存储为一个脚本文件，以便于以后使用。

讨论

R软件初学者会在命令行输入一个表达式，然后查看结果。当他有信心后，为了完成更加复杂的计算，他输入一些更加复杂的表达式。很快，他会重复输入同样的（或许有些许变化）有多行的命令。

而有经验的用户则不经常输入复杂的表达式。他们可能输入同样的表达式一次或者两次，当他们意识到这些命令有用并且会重用时，他们就会把这些命令复制、粘贴到用户界面中的脚本编辑器窗口中。在以后需要使用部分命令时，他们会在编辑器窗口中选中这部分命令，让R来执行。这就避免了再次输入这些命令。如果他们保存的代码片段有很长的代码时，这种技巧尤其有用。

在Windows和OS X操作系统中，R用户界面的一些功能支持应用这种工作方式。

打开脚本编辑器窗口

主菜单中，选择“文件”（File）→“建立新脚本”（New）。

执行脚本编辑窗口中的某一行命令

将光标定位到需要执行的那一行命令上，按Ctrl+R键执行。

执行脚本编辑窗口中的多行命令

用鼠标选中需要执行的命令，按Ctrl+R键执行。

执行脚本编辑窗口中全部的命令

按Ctrl+A键全选窗口中的所有命令，按Ctrl+R键执行。或者在主菜单中选择“编辑”（Edit）→“运行所有”（Run all）。

从命令控制台窗口中把命令复制到脚本编辑窗口中只需通过简单的复制和粘贴就可以。

退出R软件时，它会询问是否保存该程序脚本，你可以选择保存以备后用，或者放弃保存该脚本文件。

我最近应用该技巧为学生演示了中心极限定理。我希望把总体的概率密度曲线重置到样本均值的密度曲线上。

我打开R软件，然后定义变量，执行函数来绘制总体概率密度图。之后我意识到需要重复执行其中的部分命令，因此我打开R的脚本编辑窗口，并将这些命令复制过去。如图2-1所示。

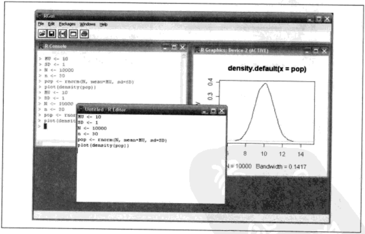


图2-1：将命令放到R的一个新编辑窗口中

我继续回到R的命令控制窗口中输入命令，期望用样本均值的密度曲线置于刚才的总体密度曲线图形上，然而第二幅密度函数图不能恰好地放置在图形中，因此我需要修改先前的函数调用，并重新执行上面的所有过程。

复制命令到编辑器窗口的好处就大大的显现出来了：我重新输入了几条命令，或者没有重新输入命令。而是把多一些的命令复制到编辑器窗口，修改对函数`plot`的调用，随后重新执行编辑器窗口中的所有命令。

此时，刚才的问题得到了解决。我对编辑器中的代码进行修改（添加标题和标签），并再次执行编辑器窗口的所有命令，得到的最终结果如图2-2所示。当上述一切完成之后，我将R编辑器窗口的内容保存为一个脚本文件，以便于随时重新绘制这幅图形。

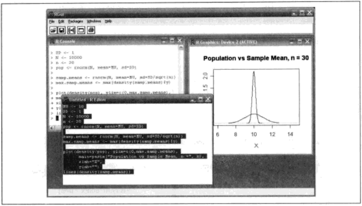


图2-2：本节描述的最终R编辑器窗口中的代码

2.14 常见错误

问题

你希望避免R初学者常犯或者某些有经验的用户也会犯的错误。

讨论

下面是一些容易给你带来麻烦的地方：

调用函数时忘记添加括号

在调用R函数时需要在函数名后加上一对圆括号。例如，以下调用ls函数：

```
> ls()
[1] "x" "y" "z"
```

然而，如果在调用过程中忽略了其后的括号，R软件则不会执行该函数。此时，R软件会给出该函数的定义，这大部分情况下不是你想要的：

```
> ls
function (name, pos = -1, envir = as.environment(pos), all.names = FALSE,
    pattern)
{
    if (!missing(name)) {
        nameValue <- try(name)
        if (identical(class(nameValue), "try-error")) {
```

```
        name <- substitute(name)
    .
    . (etc.)
    .
```

在输入Windows系统的文件路径时忘记应用双反斜杠

例如读取本地Windows系统中路径为*F:\research\bio\assay.csv*的文件。下面命令就无法执行：

```
> tbl <- read.csv("F:\research\bio\assay.csv")
```

字符串中的反斜杠（\）具有特殊含义，因此在此处需要双反斜杠。否则，R会将该路径理解为*F:researchbioassay.csv*，这不是用户所期望的。解决方案参见方法4.5。

将"<-"误写为"<（空格）-"

赋值符号为<-，在<与-之间没有空格：

```
> x <- pi        # 将3.1415926...赋值于x
```

若不小心在之间添加了空格，则表达式的意义就会完全不同：

```
> x < - pi       # 我们错误地将x与-pi进行比较，而不是对x赋值
```

上述表达式变为对x与负π(- pi)的比较，它并不改变x的值。运气好的话，x未被定义，R软件会发出提示信息告知你未找到x变量：

```
> x < - pi
Error: object "x" not found
```

如果x已经定义，则R会进行比较并返回一个逻辑值，TRUE或者FALSE。这样的输出结果也会使你意识到出现了错误，因为赋值表达式一般不会输出任何东西：

```
> x <- 0                  # 将0赋值给x
> x < - pi                # 输入错误
[1] FALSE
```

错误地跨行输入表达式

R软件会读取用户输入的内容，不管输入多少行，它会读入直到你完成一个完整的表达式。它会用提示符+提示你继续输入，直到它认为表达式已经完整。下面是将一个表达式分为两行书写的例子：

```
> total <- 1 + 2 + 3 +    # 在下一行继续输入
>   4 + 5
> print(total)
[1] 15
```

问题在于跨行输入的过程中，R软件误以为上一行的表达式已完成，这种情况很容易发生：

```
> total <- 1 + 2 + 3          # R错误地认为表达式已完成
> + 4 + 5                     # 第二行已是一个新的表达式，R对其输出结果
[1] 9
> print(total)
[1] 6
```

有两个地方可以让你认识到不妥之处：一是R软件给出了通常的提示符（>）而不是跨行继续输入的提示符（+）；二是R输出了4 + 5的结果。

这个问题给一般用户带来麻烦，而对程序员来说更是如同噩梦一般，因为这会导致R程序中难以发现的错误。

误将=用做==

双等于号（==）作为逻辑运算符用于比较。如果不小心误写成单个等号（=），则会导致不可逆地覆盖了原来的变量值：

```
> v == 0  # 将v的值与0进行比较
> v = 0   # 将0赋值给变量v，覆盖了原先v的值
```

将1:(n+1)误写为1:n+1

你可能会以为表达式1:n+1会生成1, 2, ..., n, n + 1的数列，但结果并非如此。该表达式会给出1, 2, ..., n数列的每个元素加1的数列，即数列2, 3, ..., n, n +1。原因是R软件将1:n+1理解为(1:n)+1。因此在输入时需要使用括号来得到你所希望的结果：

```
> n <- 5
> 1:n+1
[1] 2 3 4 5 6
> 1:(n+1)
[1] 1 2 3 4 5 6
```

循环规则导致的错误

向量计算和向量比较中，当两向量的长度相同时计算不会出错。然而，当参与运算的两个向量的长度不同时，可能会得到令人困惑的结果。为了防止出现这种可能性，需要了解并熟记循环规则，具体细节请参见方法5.3。

安装了软件包但没有使用library()或require()命令载入该软件包

安装软件包只是使用它的第一步，还需要后续的一个步骤：使用library或require命令将软件包载入到R的搜索路径中。没有这一步，R软件不会识别该软件包中的数据和函数。详情请参见方法3.6。

```
> truehist(x,n)
Error: could not find function "truehist"
> library(MASS)      # 将MASS数据包载入R软件中
> truehist(x,n)
>
```

误将aList[[i]]写为aList[i]，或者反之

若变量lst为一个列表，则可以采取两个方法引用其中的数据：lst[[n]]表示列表

中的第*n*个元素；而lst[n]表示仅含有*lst*的第*n*个元素的一个列表。因此两种方法差异甚大。参见方法5.7。

误将&与&&混淆，或者将|与||混淆

&与|用在含有逻辑值TRUE和FALSE的表达式中。参见方法2.9。

&&与||用在if语句或while语句中的流控制表达式中。其他编程语言的程序员会习惯性地在所有场合使用&&与||符号，因为它们运算速度快。但当用于向量或者逻辑值时，这两个符号则可能会给出意外的结果。因此除非那是你真正需要的，否则避免使用这两个运算符。

对单一参数的函数给出多个参数

对于函数mean(9,10,11)，你认为输出的结果是什么？结果不是10而是9。函数mean仅计算它的第一个参数的均值，而将第二、第三个参数视为其他位置的参数。

有些函数，如mean，仅计算一个参数。而其他的函数，如max和min，则会考虑式中所有的参数，在所有参数上应用函数。在使用前，应先对所使用的函数有所了解。

误将max与pmax混淆，或者将min与pmin混淆

max和min具有多个参数，且返回单一值：即所有参数中的最大值或最小值。

pmax与pmin有多个参数，但返回值是一个向量，它取各个参数相应元素的最大值或最小值。参见方法12.9。

用某个无法识别数据框类型的函数处理数据框数据

有些函数能够灵活地识别数据框类型的数据。它们会对数据框中各列数据分别进行计算，并给出各自的结果。如函数mean与sd就是这样的函数。这两个函数之所以能计算各列数据的均值与方差，是因为它们会将数据框中各列数据识别为不同的变量，而不是将所有数据混合在一起进行计算。

可惜的是，并非所有的函数都能这样聪明，例如函数meadian、max、min等。这些函数会将所有列的数据放到一起，并对该整体进行计算来得到结果，这可能不是你期望的。因此，在使用函数时应了解该函数是否能处理数据框数据。

未检索问题是否有答案前，就在邮件列表中张贴问题

不要浪费你的时间，也不要浪费他人的时间。在向邮件问答列表或者Stack Overflow网站上张贴问题前，请务必搜索他人的解答记录。一般情况下，你的问题都已经有过解答。因此，你可以通过阅读该问题的讨论来获得帮助。详情参见方法1.12。

另请参阅

请参见方法1.12、方法2.9、方法 5.3以及方法5.7。

第3章

R软件导览

简介

R是一个史无前例的大型软件。和应用其他大型软件一样，你需要花时间对它进行：配置、客户化、更新，以及把它融入你的计算机环境中。本章将帮助你完成这些任务。本章内容将不涉及算法、统计或者绘图，它仅仅介绍如何处理作为软件的R。

3.1 获取和设定工作目录

问题

你希望改变工作目录，或者希望了解当前工作目录。

解决方案

命令行

使用getwd函数来显示当前工作目录，使用setwd函数改变当前目录：

```
> getwd()
[1] "/home/paul/research"
> setwd("Bayes")
> getwd()
[1] "/home/paul/research/Bayes"
```

Windows

从主菜单中选择“文件”（File）→“改变工作目录”（Change dir...）。

OS X

从主菜单中选择“杂项”（Misc）→“改变工作目录”（Change Working Directory）。

对于Windows系统和OS X系统来说，菜单会以文件浏览窗口的形式打开当前目录，需要改变工作目录时则可以浏览一个新的工作目录即可。

讨论

工作目录很重要，因为它是所有输入和输出文件的默认位置，包括读取和写入数据文件、打开和保存脚本文件，以及保存工作空间的镜像。当使用R软件直接打开文件，而未指定其绝对路径时，软件会将该文件所在路径视为你的工作目录。当你没有指定绝对路径来打开一个文件时，R假定该文件位于当前工作目录。

初始的工作目录取决于你如何启动R，具体参见方法1.2。

另请参阅

有关Windows系统下如何处理文件名，参见方法4.5。

3.2 保存工作空间

问题

你希望在不退出R的情况下保存你的工作空间。

解决方案

使用save.image函数：

```
> save.image()
```

讨论

工作空间中保存有当前R的变量和函数，并且工作空间是在启动R时自动创建的。工作空间存储在计算机内存中，直到退出R软件，在退出R时可选择对其进行保存。

然而，你可能希望在不退出R软件的情况下保存工作空间，比如你希望在吃午餐前保存工作空间，防止偶然的电源或电脑故障导致数据丢失。此时可使用save.image函数。

工作空间存放于当前工作目录下的一个后缀名为*.RData*的文件中。当启动R时，计算机会寻找该文件，找到后用它初始化工作空间。

可惜的是，工作空间不会保存当前打开的图形，退出R软件后这些图形就会消失，并且

也没有简单的方法来保存和恢复这些图形。因此在退出R前，保存数据和绘制图形的R代码。

另请参阅

有关退出R时保存工作空间的细节，参见方法1.2；有关设定工作目录的细节，参见方法3.1。

3.3 查看历史命令记录

问题

你希望查看近期使用过的一系列命令。

解决方案

使用上行箭头或Ctrl+P键，能调回之前的命令。或者应用函数history来查看最近输入的命令：

```
> history()
```

讨论

history函数会显示最近使用的命令，默认显示最新的25条，当然你可以要求显示更多的历史命令：

```
> history(100)         # Show 100 most recent lines of history
> history(Inf)         # Show entire saved history
```

对于那些最近的命令而言，只需通过R的命令行编辑特性就能找回，按上行箭头或Ctrl+P键可以找回之前的命令，每按一次显示一行。

即使在退出R后，也可以查看历史命令记录。R会根据用户需求将命令记录在当前工作目录下的一个后缀名为*.Rhistory* 的文件中。可以用文本编辑器打开该文件，浏览到文件的底部，便是最近输入的命令。

3.4 保存先前命令产生的结果

问题

在R中键入一个表达式得到一个计算结果，却忘了将该结果保存到一个变量中。

解决方案

R中有一名为.Last.value的特殊变量，它存储最近一个计算出的表达式值。在输入其他内容前，可以将该特殊变量保存到其他变量中。

讨论

有时在输入一段长表达式，或者调用一个运行时间很长的函数后，忘记保存计算结果。这种情况往往很让人沮丧。幸运的是，你不需要重新输入这个表达式或重新调用函数，因为之前的运算结果已保存于.Last.value变量中。

```
> aVeryLongRunningFunction()       # Oops! Forgot to save the result!
[1] 147.6549
> x <- .Last.value                 # Capture the result now
> x
[1] 147.6549
```

需要注意的是，每次输入新的表达式后，.Last.value的值都会被改写。因此需要立即对它的值进行保存。若在输入新的表达式后才想起要保存先前的结果，那时就太晚了。

另请参阅

有关找回命令记录，参见方法3.3。

3.5 显示搜索路径

问题

你希望查看当前R中已载入哪些软件包。

解决方案

使用没有参数的search函数：

```
> search()
```

讨论

搜索路径是指当前已载入内存中的R软件包列表。虽然计算机中已安装了许多R软件包，但在使用R的某个时刻，可能仅有少数几个R包被载入了R的解释器中。因此你可能希望了解当前哪些R包已载入。

没有参数的search函数将返回已载入的R包名称列表。生成的输出类似于：

```
> search()
[1] ".GlobalEnv"        "package:stats"     "package:graphics"
[4] "package:grDevices" "package:utils"     "package:datasets"
[7] "package:methods"   "Autoloads"         "package:base"
```

不同机器安装的R包不同，因此返回的结果也可能不同。search函数返回的结果是一个字符串向量。其中，第一个字符串为".GlobalEnv"，它指你的工作空间。大多数字符串为如"package:*packagename*"的形式，表示名为*packagename*的R包当前已加载到R中。在该例中，已载入的R包有：stats、graphics、grDevices、utils等。

R通过搜索路径寻找函数。当你输入函数名称时，R会在路径中依次进行搜索，直至在某一个载入的R包中找到该函数。若找到该函数，则R执行它；否则，R显示错误信息并停止。（另外需要注意的是，搜索路径中也包含环境，而不只有R软件包。用R软件包中的对象初始化时，搜索算法也会有所不同，详见《R Language Definition》。）

由于工作空间（.GlobalEnv）是搜索列表中的第一项，所以R会首先在工作空间中寻找函数，然后才寻找其他R包。如果工作空间和其他R包中都包含某个同名的函数，则工作空间会"遮盖"后者中的函数，即R会在找到工作空间中的函数后停止搜索。因此永远无法找到其他R包中的那个函数。如果你希望重载R包中的函数，这将是一个好消息；但如果你希望使用该R包的函数时，这一特性将成为你的障碍。

R也会通过类似的方法在搜索路径中寻找数据（而非文件）。

对于UNIX用户来说，请勿将R的搜索路径与UNIX的搜索路径（环境变量PATH）混淆。两者概念上类似但却是根本不同的东西。R搜索路径是仅在R软件内部起作用，仅仅用于定位R函数和数据；而UNIX搜索路径是在UNIX系统中搜索可执行程序。

另请参阅

关于如何在R中载入R包，参见方法3.6；关于如何查看R的所有包列表（而不仅仅是已载入的R包），参见方法3.8；关于如何在搜索路径中插入数据框，参见方法5.31。

3.6 使用R包中的函数

问题

计算机中已安装的R包可以是R的标准包，或者是通过网络下载的包。当你希望使用某个R包中的函数时，R却不能找到该函数。

解决方案

使用函数library或者函数require把需要的R包载入R中：

```
> library(packagename)
```

讨论

R自带一些标准包。但在启动R时，并非所有的R包都会自动载入。同样，你可以通过CRAN网站下载和安装许多有用的R包，但是在启动R时这些包不会自动载入。例如，R包MASS是R的一个标准包，你在调用该包中的lda函数时，可能会出现以下信息：

```
> lda(x)
Error: could not find function "lda"
```

R显示无法在当前载入内存中的R包中找到lda函数。

当使用library函数或require函数时，R将该对应的包载入内存中，此时载入的包中的内容便是可以使用的：

```
> lda(f ~ x + y)
Error: could not find function "lda"
> library(MASS)                          # Load the MASS library into memory
> lda(f ~ x + y)                         # Now R can find the function
Call:
lda(f ~ x + y)

Prior probabilities of groups:
.
. (etc.)
.
```

在调用library函数前，R无法识别函数名。用library函数载入R包后，该R包中的内容便可调用，此时可正常使用lda函数。

注意，在library函数中的R包名称不需要用双引号括起来。

require函数与library函数几乎类似，但require函数有两个特性便于脚本程序的书写。一是它在成功载入R包后，返回TRUE；否则，返回FALSE。二是在载入失败时，它仅仅给出一条警告信息；而library载入失败时，会生成错误信息。

两个函数都有一个共同特性，即它们不会重复载入已载入的R包。因此可以两次重复调用同一个R包。这对书写脚本很有帮助，既然已经载入的R包不会再次载入，脚本就可以放心载入所需要的R包，而不用担心它是否已载入。

detach函数会卸载当前已载入的R包：

```
> detach(package:MASS)
```

注意，R包名称必须符合规范，即以package:MASS 的形式出现。

卸载R包的一个原因是，该包中的函数名与搜索列表中该包后面的包的某函数重名。发生这种情况时，次序在前的函数（搜索列表靠前的包中的函数）会遮盖次序在后的函数（搜索列表靠后的包中的函数）。R在找到高次序的函数后便会停止搜索。因此，你无法看到低次序的函数，所以需要卸载高次序的R包来使用低次序的包中的函数。

另请参阅

参见方法3.5。

3.7 使用R的内置数据集

问题

你希望使用R的内置数据集。

解决方案

由于搜索路径中已包含datasets包，所以R的标准数据集是可以使用的。

要调用其他R包中的数据集，使用data函数并在参数中给出相应的包名称和数据集名称：

```
> data(dsname, package="pkgname")
```

讨论

R中包含许多内置数据集。由于R提供了这些数据集，你可以应用它们来进行试验，所以它们对学习和使用R很有帮助。

许多数据集都包含在名为datasets的R包中。该R包是R的基础包，它在R的搜索路径中，因此可以直接应用这些数据集。例如，可以调用内置数据集pressure：

```
> head(pressure)
  temperature pressure
1           0   0.0002
2          20   0.0012
3          40   0.0060
```

```
4          60  0.0300
5          80  0.0900
6         100  0.2700
```

你如果希望了解更多有关数据集pressure的信息，可以使用help函数来学习该数据集或者其他数据集：

```
> help(pressure)          # Bring up help page for pressure dataset
```

你也可以不加参数地使用data函数，它返回datasets包内容的一个列表。

```
> data()                  # Bring up a list of datasets
```

任何R包都可以选择包含数据集，来补充R包datasets中的数据。例如，MASS包中包含了许多有趣的数据集。使用带有package参数的data函数可以访问相应的R包中的数据集。MASS包中包含数据集Cars93，可通过以下方法来访问：

```
> data(Cars93, package="MASS")
```

这样调用了data函数后，便可随时使用Cars93数据，并可执行如summary(Cars93)、head(Cars93)等命令。

当把R包加入到搜索路径中后（例如通过library(MASS)），你就不需要调用函数data来访问该包中的数据集了。因为在加载该R包后，它的数据集便自动可用。

可以使用data函数并把R包的名称作为package参数的值（不需要数据集名称参数），来查看MASS或其他包中的数据集列表：

```
> data(package="pkgname")
```

另请参阅

有关搜索路径的更多详情，参见方法3.5；有关R包和library函数，参见方法3.6。

3.8 查看已安装的R包列表

问题

你希望了解哪些R包已安装到你的计算机中。

解决方案

调用没有参数的library函数来查看基本的R包列表。使用installed.packages 命令查看更多R包的细节信息。

讨论

不含参数的library函数会显示已安装的R包列表，这一列表可能会很长。在Linux系统中，输出结果的前几行可能会显示如下：

```
> library()
Packages in library '/usr/local/lib/R/site-library':

boot                 Bootstrap R (S-Plus) Functions (Canty)
CGIwithR             CGI Programming in R
class                Functions for Classification
cluster              Cluster Analysis Extended Rousseeuw et al.
DBI                  R Database Interface
expsmooth            Data sets for "Forecasting with exponential
                     smoothing"
.
. (etc.)
.
```

在Windows和OS X系统中，会弹出一个新窗口显示结果。

可以使用installed.packages函数获得更多信息，该函数会以矩阵的形式显示计算机中已安装的R包的信息。矩阵每行对应一个已经安装的R包，矩阵的列表示R包名称、搜索路径、版本等信息。这些信息都取自已经安装的R包的内部数据库。

通过通常的索引方法，可以从该矩阵中得到有用的信息。下面的Windows系统中的代码通过调用installed.packages函数，提取了Package和Version两列，可以清晰地看到已安装的R包的版本。

```
> installed.packages()[,c("Package","Version")]
           Package      Version
acepack    "acepack"    "1.3-2.2"
alr3       "alr3"       "1.0.9"
base       "base"       "2.4.1"
boot       "boot"       "1.2-27"
bootstrap  "bootstrap"  "1.0-20"
calibrate  "calibrate"  "0.0"
car        "car"        "1.2-1"
chron      "chron"      "2.3-12"
class      "class"      "7.2-30"
cluster    "cluster"    "1.11.4"
.
. (etc.)
.
```

另请参阅

有关如何将R包载入计算机内存，参见方法3.6。

3.9 从CRAN网站安装R包

问题

你在CRAN网站中找到某R包，希望下载并安装到本地计算机中。

解决方案

命令行

使用install.packages函数，在函数参数中键入R包名称并用引号括起：

```
> install.packages("packagename")
```

Windows

在主菜单中选择"软件包"（Package）→"安装软件包"...（Install package(s)）进行安装。

OS X

在主菜单中选择"软件包与数据"（Package & Data）→"安装软件包"(Package Installer)进行安装。

对于所有的平台，安装前需要先选择CRAN网站的镜像。

在Windows和OS X上，系统会要求用户选择需要下载的R包。

在Linux或UNIX中的系统级安装

因为在Linux或UNIX中，这些安装路径通常不允许一般用户进行读、写，所以如果是对这些R包进行系统级的安装，则需要有root权限。因此，安装的工作一般都需要系统管理员来完成。如果系统管理员不愿意或无法进行系统级的安装，你也可以将R包安装至你的个人目录中。

如果你拥有root权限：

1. 使用su或sudo命令启动root界面。
2. 在root界面中开启R进程。
3. 在此处执行install.packages函数

> 如果你不具有root权限，你将很快就会知道。install.packages函数会停止运行，并警告你无法在必需的目录中写入数据。此时它会询问是否要将文件创建于个人的目录中。如果选择是，它会在你的个人目录下创建必要的目录，并在其中安装相应的R包。

讨论

在本地安装软件包，是使用程序的第一步。安装时需要选择一个镜像站点来下载这些软件包文件：

```
--- Please select a CRAN mirror for use in this session ---
```

之后它会显示一个CRAN镜像站点列表，选择离你最近的站点。

CRAN的官方服务器是一个相对一般的机器，它位于奥地利的维也纳市的维也纳经济学院的统计和数学学院（WU Wien, Vienna，Austria）。如果每个R软件用户都从该服务器下载文件，那么该服务器势必会超载，因此在全球各地设置了多个镜像站点。强烈建议使用离你最近的镜像服务器进行下载。

如果新的R包需要其他R包的支持，而本地计算机之前未曾安装。那么R安装程序会自动下载安装这些需要的R包。这一功能避免了用户去查询各个R包之间是否关联。

在Linux或UNIX系统中安装时要做一个特殊的考虑。你可以选择在系统目录中或者个人目录中安装该程序。系统目录中的程序所有用户都可以使用，而个人目录中的程序（一般）仅个人能使用。因此对于广泛应用的R包可以设定安装到系统目录中，而对于那些个人的或未测试的R包则可以安装到个人目录中。

默认情况下，install.packages函数假定你需要安装到系统目录中。若需要安装至个人目录中，首先建立一个目录，例如*~/lib/R*：

```
$ mkdir ~/lib
$ mkdir ~/lib/R
```

启动R前先设定环境变量R_LIBS。否则，R软件无法得知你的个人目录地址：

```
$ export R_LIBS=~/lib/R        # bash syntax
$ setenv R_LIBS ~/lib/R        # csh syntax
```

之后，调用install.packages函数并设定lib参数来选择安装目录：

```
> install.packages("packagename", lib="~/lib/R")
```

另请参阅

有关如何寻找相关R包，参见方法1.11；有关安装后如何使用R包，参见方法3.6。

3.10 设定默认CRAN网站镜像

问题

你希望下载某些R包，因此希望设定默认的CRAN网站镜像，这样R每次下载时不需要你选择镜像。

解决方案

该方案要求用户R系统中包含一个*.Rprofile* 文件，如方法3.16描述的那样：

1. 调用chooseCRANmirror函数：

```
> chooseCRANmirror()
```

R会显示CRAN镜像的列表。

2. 从列表中选择镜像并点击确定。

3. 通过查看repos选项的第一个元素来获取所选镜像的URL地址：

```
> options("repos")[[1]][1]
```

4. 将下面的命令添加至*.Rprofile* 文件中：

```
options(repos="URL")
```

其中的URL就是镜像的URL。

讨论

在每次安装R包的过程中都会使用相同的CRAN镜像（即离本地最近的镜像地址）。你可能对于R重复地询问选择镜像感到厌烦。按照上述方法所给出的解决方案进行操作，便设定了默认的镜像，R每次就不再询问了。

repos选项是默认镜像的名称。使用chooseCRANmirror函数选择镜像时会有一个重要的副作用，即按照选择来设定repos选项。问题是当R退出时，R不会保存选择的镜像为默认镜像。通过在*.Rprofile*中对repos进行设定，R在启动时会自动恢复你的设定。

另请参阅

有关*.Rprofile*文件与options函数的详情，参见方法3.16。

3.11 隐藏启动信息

问题

你不希望显示冗长的R启动信息。

解决方案

在启动R时，在命令行中添加--quiet选项。

讨论

对于初学者而言，R启动信息包含了R软件和获取帮助的有用信息。但这种信息的新奇感将很快消失。

若通过命令行启动R，则可以使用--quiet选项隐藏启动信息：

```
$ R --quiet
>
```

如在我的Linux计算机中，我对R进行如下的命名，从而隐藏启动信息：

```
$ alias R="/usr/bin/R --quiet"
```

若在Windows系统中通过快捷方式启动R，可以在R快捷方式中添加--quiet选项，隐藏启动信息。右击快捷方式图标，选择“属性”（Properties），选择“快捷方式”（Shortcut）标签，在“目标”（Target）字符串的结尾，添加--quiet选项。注意，在程序路径和--quiet之间要留下空格。

另请参阅

参见方法1.2。

3.12 运行脚本

问题

你希望运行那些已保存到文本文件中的R代码。

解决方案

source函数使R读取文本文件并执行其内容：

```
> source("myScript.R")
```

讨论

建议将大篇幅或常用的R代码存储于文本文件中，这样能免于重复输入同样的代码。使用source函数读取并运行这些代码，它等同于将这些代码键入R控制台。

假设文件*hello.R*中包含如下代码：

```
print("Hello, World!")
```

读取并执行该文件的内容：

```
> source("hello.R")
[1] "Hello, World!"
```

设定参数echo=TRUE，使R在运行这些代码前显示每一行带有R提示符的代码命令：

```
> source("hello.R", echo=TRUE)

> print("Hello, World!")
[1] "Hello, World!"
```

另请参阅

如何在用户界面运行R代码块，参见方法2.13。

3.13 批量运行R代码

问题

你正在编写一个命令脚本，如UNIX或OS X系统的shell脚本，或Windows系统中的批处理脚本，并且你希望在这些脚本中执行R代码。

解决方案

使用带有CMD BATCH子命令的方式运行R程序，并给出脚本文件名和输出文件名：

```
$ R CMD BATCH scriptfile outputfile
```

如果需要将输出结果发送到标准输出设备中，或者希望将命令行参数传递到脚本中，可以考虑应用Rscript命令：

```
$ Rscript scriptfile arg1 arg2 arg3
```

讨论

R是一个交互式软件，它提示用户输入，然后显示输出结果。有时你想在批处理模式下运行R，从脚本读取命令。这对于shell内部的脚本，例如含有统计分析的脚本而言特别有用。

CMD BATCH子命令把R转到批处理模式，它读取脚本文件*scriptfile*并且把输出写入输出文件*outputfile*。这个运行过程中不与用户交互。

你可能会根据具体情况使用命令行选项，调整R的批处理过程。例如，使用--quiet选项来避免启动信息，否则将使输出信息混乱：

```
$ R CMD BATCH --quiet myScript.R results.out
```

下面是一些其他批处理模式下的实用命令：

--slave
: 类似于--quiet,它禁止回送输入的信息，使R软件输出的信息更为简洁。

--no-restore
: 在R启动时不还原工作空间。对于希望以空白工作空间启动R的脚本而言，这个选项很有必要。

--no-save
: 在退出R时，不保存工作空间；否则，R会保存当前工作空间并覆盖原有工作目录中的*.RData*文件。

--no-init-file
: 不读取*.Rprofile*文件或者~/. *Rprofile*文件。

在脚本运行结束后，CMD BATCH子命令一般会使用proc.time函数显示其执行的时间。如果你不需要显示该时间，可以在代码最后一行调用参数为runLast=FALSE的q函数，它将防止调用proc.time函数。

CMD BATCH子命令有两个限制条件：输出必须总是传送到一个文件中，并且无法简单地将命令行参数传送到脚本中。如果这两个限制成为问题，可以考虑使用R软件自带的Rscript程序。Rscript命令的第一个命令行参数是脚本文件的名称，其余的参数将传递给脚本代码：

```
$ Rscript myScript.R arg1 arg2 arg3
```

在脚本中，命令行参数可以通过调用commandArgs函数来获取，该函数会把参数作为一个字符串向量返回：

```
argv <- commandArgs(TRUE)
```

Rscript程序和CMD BATCH命令使用上面所提到的相同的命令行选项。

将输出结果输出到标准输出设备中，该设备是R从调用它的shell脚本中继承来的。当然，可以通过一般的重定向方法将输出重定向到一个文件中：

```
$ Rscript --slave myScript.R arg1 arg2 arg3 >results.out
```

下面是一个名为*arith.R*的简易R脚本文件，它对两个命令行参数进行四个算术运算：

```
argv <- commandArgs(TRUE)
x <- as.numeric(argv[1])
y <- as.numeric(argv[2])

cat("x =", x, "\n")
cat("y =", y, "\n")
cat("x + y = ", x + y, "\n")
cat("x - y = ", x - y, "\n")
cat("x * y = ", x * y, "\n")
cat("x / y = ", x / y, "\n")
```

脚本以下面的形式调用：

```
$ Rscript arith.R 2 3.1415
```

产生如下结果：

```
x = 2
y = 3.1415
x + y = 5.1415
x - y = -1.1415
x * y = 6.283
x / y = 0.6366385
```

在Linux或UNIX系统中，你可以在脚本的开头添加#!后跟随Rscript程序的路径，这样脚本就是完全自我包含的了（即代码变得完全独立于外部）。假定Rscript程序安装在*/usr/bin/Rscript*目录中，你可以在*arith.R*脚本文件中添加下面一行，使其成为自我包含代码：

```
#!/usr/bin/Rscript --slave
```

```
argv <- commandArgs(TRUE)
x <- as.numeric(argv[1])
.
. (etc.)
.
```

在提示符处，我们把脚本文件标记为可执行文件：

```
$ chmod +x arith.R
```

此时我们可以不用Rscript前缀而直接调用脚本代码：

```
$ arith.R 2 3.1415
```

另请参阅

有关R中运行脚本文件的详情，参见方法3.12。

3.14 获取和设定环境变量

问题

你希望查看某个环境变量的值，或者改变其值。

解决方案

使用Sys.getenv函数查看环境变量的值，使用Sys.putenv函数改变环境变量的值：

```
> Sys.getenv("SHELL")
       SHELL
"/bin/bash"
> Sys.setenv(SHELL="/bin/ksh")
```

讨论

环境变量常在UNIX系统中用于设定或控制软件的运行。每个进程都有继承于其父进程的环境变量。为了理解R程序的运行过程，有时希望查看R进程环境变量的设定。同样，有时也需要改变环境变量的设定来改变R的运行过程。

我有时在某处打开R，但希望将图像显示于其他地方。例如，有时我在Linux系统的一个终端中运行R，但希望将图形显示在一个大屏幕中，这样便于观众看得更加清楚。或者，我在我的Linux工作站中运行R，希望将图形结果显示在同事的工作站中。在Linux系统中，R用X Windows系统来显示图形。X Windows按照名为DISPLAY的环境变量来选择显示设备。可以通过Sys.getenv函数查看环境变量DISPLAY的取值：

```
> Sys.getenv("DISPLAY")
DISPLAY
  ":0.0"
```

所有环境变量的值都是字符串形式的，例如此处的值为":0.0"，这代表R会话连接到了工作站的“0号显示器，0号屏幕”的显示接口。

为了将图形重定向到本地显示接口10.0，可以对DISPLAY参数做如下修改：

```
> Sys.putenv(DISPLAY="localhost:10.0")
```

同样，可以将图形重定向到名为zeus的工作站的“0号显示器，0号屏幕”的显示接口。

```
> Sys.putenv(DISPLAY="zeus:0.0")
```

对于上述两种情况而言，必须在绘制图形前改变DISPLAY变量。

3.15 找到R的主目录

问题

你需要了解R的主目录，即所有设置文件与安装文件放置的目录。

解决方案

R生成一个名为R_HOME的环境变量，可以通过Sys.getenv函数查看：

```
> Sys.getenv("R_HOME")
```

讨论

大多数用户不需要知道R的主目录。但系统管理员或高级用户必须知道，以便于管理或改变R的安装文件。

在R启动时，会定义一个名为R_HOME的环境变量（不是R变量），此变量是R主目录的路径地址。可以通过Sys.getenv函数查看其值。下面是各个不同操作系统平台的示例。对于不同的计算机其返回的值也必然有所不同：

Windows

```
> Sys.getenv("R_HOME")
                    R_HOME
"C:\\PROGRA~1\\R\\R-21~1.1"
```

OS X

```
> Sys.getenv("R_HOME")
"/Library/Frameworks/R.framework/Resources"
```

*Linux*或*UNIX*

```
> Sys.getenv("R_HOME")
     R_HOME
"/usr/lib/R"
```

Windows系统中的结果看上去似乎很古怪，原因是R返回的是旧的DOS形式的紧凑形式的路径名。这里，完整的用户友好的地址名应为*C:\Program Files\R\R-2.10.1*。

在UNIX和OS X操作系统中，可以从用户终端运行R软件，并使用RHOME子命令显示主目录地址：

```
$ R RHOME
/usr/lib/R
```

注意，R在UNIX和OS X系统的主目录中包含安装文件，但不一定包含R的可执行文件。例如，R的主目录为*/usr/lib/R*，而可执行文件可能存放于*/usr/bin*目录中。

3.16 R的客户化

问题

你希望通过改变配置选项或预加载R包，来客户化R进程。

解决方案

建立名为*.Rprofile*的脚本文件来对R进程进行客户化，R会在启动时执行该文件。不同操作系统的*.Rprofile*文件的存放位置有所不同：

OS X、*Linux*或*UNIX*

文件保存于主目录中（*~/.Rprofile*）。

Windows

文件保存于“我的文档”（My Document）中。

讨论

R在启动时会执行配置脚本文件（*.Rprofile*）。避免用户重复载入常用的R包以及重复调整R配置选项。

可以建立名为.*Rprofile*的配置脚本文件，并将其存放于主目录中（OS X、Linux和UNIX操作系统），或者“我的文档”（My Document）目录中（Windows XP），或者“文档”（Documents）目录中（Windows Vista和Windows 7）。配置文件.*Rprofile*中可以使用函数来对你的进程进行客户化，例如下面的简单脚本将载入MASSR包，并且将R命令提示符改为R>：

```
require(MASS)
options(prompt="R> ")
```

配置文件.*Rprofile*在非常基础的程序环境下运行，因此它所能配置的功能有限。例如，在配置文件中试图打开绘图窗口时会提示错误信息，因为绘图的软件包还没有载入。同样，在配置文件中也应避免进行复杂的计算。

你也可以在项目文件目录中添加一个.*Rprofile*文件来对某个特定的项目进行客户化。当R从该目录启动时，它读取当前目录下的.*Rprofile*文件；你可以通过这样的方法对特定项目进行客户化制定配置（例如仅载入该任务需要的R包）。然而，R在找到局部配置文件后便不再读取全局配置文件。这种情况有时会变得麻烦，但也容易解决：从局部配置文件中应用source函数来调用全局配置文件。例如，在UNIX系统中，局部配置文件将首先执行全局配置文件，然后再执行其局部的内容：

```
source("~/.Rprofile")
#
# ... remainder of local .Rprofile...
#
```

设定选项

某些客户化工作是通过调用options函数来设定配置选项来完成的。R中存在许多这样的选项，对options使用help函数可以查看这些选项的列表：

```
> help(options)
```

下面是一些例子：

`browser="path"`
: HTML浏览器的默认路径。

`digits=n`
: 返回数值型值时显示的位数。

`editor="path"`
: 默认文本处理器路径。

prompt="*string*"

输入提示符。

repos="*url*"

默认存放R包的URL地址。

warn=*n*

控制显示警告信息。

载入R包

另一个常用的客户化选项是预载入R包。你可能希望R在每次启动时载入某些R包，因为这些包需要经常使用。可以简单地通过在*.Rprofile*文件中添加require函数得以实现。例如下面的调用（对应载入tseries R包）：

```
require(tseries)
```

如果调用require时给出了警告信息，那么它们会在R启动时把你的输出信息搞乱，此时可以通过把它作为suppressMessages函数的参数而使警告信息消失：

```
suppressMessages(require(tseries))
```

除了显式地调用require函数外，也可以设定名为defaultPackages的参数完成该工作，这是R启动时自动加载的R包列表，它最初包含了系统定义的列表。如果你将R包的名称添加至列表的最后，则R会在启动时载入这些R包，省去了你自己使用library或require函数来载入这些R包。

下面是我的R配置文件中用于调整预载入R包列表的片段。由于我经常使用R的zoo包，所以将它添加到defaultpackages列表中：

```
pkgs <- getOption("defaultPackages")     # Get system list of packages to load
pkgs <- c(pkgs, "zoo")                   # Append "zoo" to list
options(defaultPackages = pkgs)          # Update configuration option
rm(pkgs)                                 # Clean up our temp variable
```

使用defaultPackages的一个特点在于你可以控制R包在搜索列表中的排列顺序。添加在最后的R包最后才会被载入。因此在上例中将zoo包添加到搜索列表的开头（记住，R包在载入时将被添加到搜索列表的开头）。

预载入zoo包的代价是R在启动时会有点慢，即便有时候我不需要使用zoo包。对我来说，由于我不希望每次启动R时都输入library(zoo)函数，所以虽然R启动慢了些，但它带来的方便还是值得这样做的。

启动顺序

下面简单介绍R启动时的一系列过程（使用help(Startup)命令查看详细信息）：

1. R执行*Rprofile.site*中的脚本。这个脚本文件是系统级的脚本，它允许系统管理员对默认选项进行自定义修改。该代码文件的完整路径为R_HOME/*etc*/*Rprofile.site*（其中，R_HOME是R的主目录，参见方法3.15）。

   R发行版中不包含*Rprofile.site*文件，所以系统管理员可以根据需要自行建立该文件。

2. R执行工作目录中的*.Rprofile*脚本文件；若该文件不存在，则执行用户主目录中的*.Rprofile*文件。在这一步用户可根据自己的需要来对R进行客户化。用户主目录中的*.Rprofile*文件用于全局性的客户化。当R在低级别的目录启动时，这个低级别目录下的*.Rprofile*脚本文件也可以对在本目录下启动的R进行客户化。例如，对启动于某个项目目录下的R进行客户化。

3. 如果当前工作目录中有*.RData*文件，那么R将载入该*.RData*文件中保存的工作空间。R在退出时会将工作空间保存到一个名为*.Rdata*的文件中。它将从该文件中载入你的工作空间，并恢复访问原来的局部变量和函数。

4. 如果你定义过*.First*函数，R将执行该函数。*.First*函数是用户或项目定义启动初始化代码的好地方，你可以在*.Rprofile*文件或工作空间中来对该函数进行定义。

5. R执行*.First.sys*函数。这一步会载入默认的R包，该函数是R的内部函数，一般用户或管理员不需要对其修改。

注意，R直到最后一步执行*.First.sys*函数时才会载入默认R包。在这之前只有基础R包会载入。这一点很重要，因为它意味着之前几步不能假定除基础R包以外的软件包会载入。这也是为什么在*.Rprofile*脚本文件中试图打开绘图窗口时会出错：因为绘图R包还未载入。

另请参阅

有关载入R包的内容，参见方法3.6；关于R启动（help(Startup)）和选项（help(options)）的内容，参见R帮助页面。

第4章

输入与输出

简介

所有统计工作都是从数据开始，而大多数数据存储于文件和数据库中。对于任何大型统计项目而言，第一步可能都是处理输入。

所有统计工作最终都是将某些数值结果报告给客户，即便客户是你本人。格式化并产生输出也许是统计项目中最关键的步骤。一般的R用户可以使用简单的函数进行数据输入，如使用read.csv函数读取CSV文件，或者read.table函数读取更复杂的表格数据。也可以通过print、cat和format这样的函数给出简单的报告。

对于需要大量进行输入输出（I/O）的用户来说，强烈推荐阅读《R Data Import/Export》这本指南书，可从R的CRAN网站（*http://cran.r-project.org/doc/manuals/R-data.pdf*）下载得到。这本手册介绍了如何从电子表格、二进制文件、其他统计软件和关系型数据库中读取数据。

关于R软件原理的解释

我的一些使用SAS的朋友，对R软件的输入功能感到很失望。他们指出SAS含有一套完善的命令来读取和解析多种格式的数据文件。R软件中没有这些命令，因此他们得出R软件无法用于实际工作中，毕竟如果无法读取数据，那么如何实施工作？

我认为他们没有理解R软件的设计原理，R的设计是基于一个名为S的统计软件包。S软件的作者当时在贝尔实验室工作，他们深谙UNIX的设计理念。其设计理念的核心思想之一就是模块化工具。在UNIX系统中，一般没有体积庞大能处理所有功能的程序，而

是一些能有效处理单独任务的小程序。UNIX用户像砌砖一样将这些程序模块结合起来，形成完整的系统。

R具有强大的统计和绘图功能，它相比于其他商业软件包而言要强大得多。

然而，R却不是一个很好的处理数据文件的工具。S的作者假定用户能通过其他工具来处理数据输入的任务：perl、awk、sed、剪切、粘贴，或者其他各种可行的方法。因此为何需要重复地在R中添加这些功能呢？

如果你难以通过R来读取或解析数据，那么你可以考虑在将数据导入R前，使用其他软件对数据进行预处理，然后再让R完成它擅长的工作。

4.1 使用键盘输入数据

问题

你有少量数据，由于数据很少而无需费力创建一个输入文件，你只希望直接将数据输入到工作空间中。

解决方案

对于很小的数据集，使用c()建立向量并输入数据的内容：

```
> scores <- c(61, 66, 90, 88, 100)
```

另外，可以先建立一个空的数据框，之后再使用R内置的表格编辑器填充其中的内容：

```
> scores <- data.frame()     # Create empty data frame
> scores <- edit(score)      # Invoke editor, overwrite with edited data
```

在Windows系统中，可以在菜单中选择“编辑”（Edit）→“数据编辑器”（Data frame...）来调用数据编辑器。

讨论

在处理一些很简单的问题时，我不想麻烦地生成数据文件再用R软件读取这些文件，我只想简单地将这些数据输入到R中。最简单的方法就是用c()函数把数据构造为向量，将数据输入到R中，如解决方案中说明的那样。

通过把每个变量（列）作为一个向量，这个方法也适用于数据框：

```
> points <- data.frame(
+          label=c("Low", "Mid", "High"),
```

```
+                lbound=c( 0, 0.67, 1.64),
+                ubound=c(0.674, 1.64, 2.33)
+        )
```

另请参阅

关于如何使用R内置的数据编辑器，参见方法5.26中的解决方案。

4.2 显示更少的位数（或更多的位数）

问题

你觉得输出结果的数据有太多或太少的位数，因此你希望R显示更少或更多位数。

解决方案

print函数中的digits参数能控制显示结果的位数。

在cat函数中，使用format函数（它也包含一个digits参数）来改变输出数据的格式。

讨论

R一般以7位浮点数的形式作为输出格式：

```
> pi
[1] 3.141593
> 100*pi
[1] 314.1593
```

这对于大多数情况来说都没问题，但在很小的空间内打印大量数据的时候就会变得很麻烦。而有时结果只有几位小数，但R还是会以7位的形式输出，这种情况会导致误解。

使用print函数中的digits参数可以改变输出数据的位数：

```
> print(pi, digits=4)
[1] 3.142
> print(100*pi, digits=4)
[1] 314.2
```

cat函数不能直接控制数据的显示格式。不过，在调用cat函数前可以使用format函数先设定数据的格式：

```
> cat(pi, "\n")
3.141593
```

```
> cat(format(pi,digits=4), "\n")
3.142
```

在R中，函数print和format都会同时对整个向量一次性地进行格式化：

```
> pnorm(-3:3)
[1] 0.001349898 0.022750132 0.158655254 0.500000000 0.841344746 0.977249868
[7] 0.998650102
> print(pnorm(-3:3), digits=3)
[1] 0.00135 0.02275 0.15866 0.50000 0.84134 0.97725 0.9986
```

注意，print函数对向量元素进行一致的格式化：先找到格式化最小数值所必需的位数，然后把所有数据格式化为相同的宽度（位数并不一定相同）。这对于格式化整个表格尤其有用：

```
> q <- seq(from=0,to=3,by=0.5)
> tbl <- data.frame(Quant=q, Lower=pnorm(-q), Upper=pnorm(q))
> tbl                               # Unformatted print
  Quant       Lower       pper
1   0.0 0.500000000 0.5000000
2   0.5 0.308537539 0.6914625
3   1.0 0.158655254 0.8413447
4   1.5 0.066807201 0.9331928
5   2.0 0.022750132 0.9772499
6   2.5 0.006209665 0.9937903
7   3.0 0.001349898 0.9986501
> print(tbl,digits=2)               # Formatted print: fewer digits
  Quant Lower  Upper
1   0.0 0.5000 0.50
2   0.5 0.3085 0.69
3   1.0 0.1587 0.84
4   1.5 0.0668 0.93
5   2.0 0.0228 0.98
6   2.5 0.0062 0.99
7   3.0 0.0013 1.00
```

你也可以通过options函数设定默认的位数(digits)，它将改变所有输出结果的格式：

```
> pi
[1] 3.141593
> options(digits=15)
> pi
[1] 3.14159265358979
```

但从我的经验来看，这种方法不可取。因为它也改变了R内置函数的输出格式，而这种改变的结果可能不会令人满意。

另请参阅

其他修改格式的函数有sprintf和formatC，更多信息查看帮助页面。

4.3 将输出结果重定向到某一文件

问题

你希望将输出结果重定向到某一文件，而不是输出到R控制台。

解决方案

通过cat函数使用其file参数，可以对输出结果重定向：

```
> cat("The answer is", answer, "\n", file="filename")
```

使用sink函数对所有print和cat函数的输出结果进行重定向。在调用sink函数时，用文件名作为参数就能将控制台中的输出结果重定向到该文件。当输出完成后，可以使用不含参数的sink函数来关闭该文件，并把输出重新定向到控制台：

```
> sink("filename")                  # Begin writing output to file
. . . other session work . . .
> sink()                            # Resume writing output to console
```

讨论

函数print和函数cat一般会将结果输出到R控制台。在cat函数中可以设定file参数把输出写入某个文件，其中file参数的值可以是文件名，也可以是一个链接。print函数无法重定向它的输出，但sink函数可以强制所有输出到一个文件。一个常用的方式是用sink函数捕获R脚本的输出：

```
> sink("script_output.txt")         # Redirect output to file
> source("script.R")                # Run the script, capturing its output
> sink()                            # Resume writing output to console
```

如果重复地使用cat函数将结果重定向到文件中，那么一定要确认append参数取值为TRUE；否则，每次调用cat函数会覆盖该文件中的原有内容：

```
cat(data, file="analysisReport.out")
cat(results, file="analysisRepart.out", append=TRUE)
cat(conclusion, file="analysisReport.out", append=TRUE)
```

像上面一样对文件名进行硬编码是一个繁琐且很容易出错的过程。你是否察觉到上例第二行中文件名拼错了？与其每次重复对文件名硬编码，不如打开一个到文件的链接，并将结果输出到这一链接中：

```
con <- file("analysisReport.out", "w")
cat(data, file=con)
cat(results, file=con)
cat(conclusion, file=con)
close(con)
```

（在输出到链接时，你不需要参数append=TRUE，因为在这里这一参数设置是隐含存在的。）这一方法在R脚本中十分有用，因为这将使你的代码变得更可靠，更方便维护。

4.4 显示文件列表

问题

你希望在R中显示文件列表，而不用麻烦地转换到文件浏览器。

解决方案

函数list.files能显示当前工作目录中的文件：

```
> list.files()
```

讨论

该函数使用简单且方便。如果我不记得数据文件的名称（例如，该文件名是*sample_data.csv*，还是*sample-data.csv*）我就可以通过list.files函数快速地找到：

```
> list.files()
[1] "sample-data.csv" "script.R"
```

要查看子目录中包含的所有文件，使用list.files(recursive=T)命令。

函数list.files的一个问题在于它会忽略隐藏文件——即以句号（.）开头的文件名。如果你希望查看隐藏文件，设定函数list.files的参数all.files=TRUE即可。

```
> list.files(all.files=TRUE)
```

另请参阅

R中还包含有其他便捷处理文件的函数，参见help(files)命令输出的文档。

4.5 解决无法在Windows中打开文件的问题

问题

你在Windows系统中运行R，并想读取名为*C:\data\sample.txt*的文件。R显示无法打开该文件，但是你知道该文件确实存在。

解决方案

文件路径中的反斜杠（"\"）会导致一些问题。可以通过以下两种方法解决：

- 将反斜杠改为一般斜杠："C:/data/sample.txt"。
- 双写反斜杠："C:\\data\\sample.txt"。

讨论

当你在R中打开某一文件时，你以字符串的形式给出文件名。如果文件地址中包含反斜杠（\），则会带来问题，因为反斜杠在字符串中有特殊的含义。你可能会碰到如下情形：

```
> samp <- read.csv("C:\Data\sample-data.csv")
Error in file(file, "rt") : cannot open the connection
In addition: Warning messages:
1: '\D' is an unrecognized escape in a character string
2: '\s' is an unrecognized escape in a character string
3: unrecognized escapes removed from "C:\Data\sample-data.csv"
4: In file(file, "rt") :
  cannot open file 'C:Datasample-data.csv': No such file or directory
```

R会跳过反斜杠后的字母，然后去掉反斜杠。此时就会在上例中造成一个无意义的文件地址，如本例中的*C:Datasample-data.csv*。

使用一般的斜杠来代替反斜杠，就能简单地解决该问题。在R中斜杠没有其他特殊意义，而Windows系统将斜杠作为反斜杠来对待，这样问题就解决了：

```
> samp <- read.csv("C:/Data/sample-data.csv")
>
```

另一种解决方法是双写反斜杠，因为R软件中两个反斜杠代表一个反斜杠字符：

```
> samp <- read.csv("C:\\Data\\sample-data.csv")
>
```

4.6 阅读固定宽度数据记录

问题

你希望读取一个固定宽度数据记录的文件，即文件中的数据记录有固定的分界。

解决方案

使用read.fwf函数阅读这类文件，它的主要参数为文件名和每一栏数据的宽度：

```
> records <- read.fwf("filename", widths=c(w1, w2, ..., wn))
```

讨论

假设我们希望读取整个固定长度记录的数据文件，例如下面的*fixed-width.txt*文件：

```
Fisher    R.A.      1890 1962
Pearson   Karl      1857 1936
Cox       Gertrude  1900 1978
Yates     Frank     1902 1994
Smith     Kirstine  1878 1939
```

我们需要了解每一列的宽度。在此例中每列分别为："last name"为10个字符；"first name"为10个字符；"year of birth"为4个字符；"year of death"为4个字符。在最后两列之间有一个字符的空格。因此我们可以这样读取文件：

```
> records <- read.fwf("fixed-width.txt", widths=c(10,10,4,-1,4))
```

函数的参数widths中的-1意味着有一个1个字符的列应该被忽略，read.fwf函数的结果以数据框形式输出：

```
> records
          V1         V2   V3     V4
1 Fisher     R.A.         1890   1962
2 Pearson    Karl         1857   1936
3 Cox        Gertrude     1900   1978
4 Yates      Frank        1902   1994
5 Smith      Kirstine     1878   1939
```

注意，R会添加一些有趣的合成的列名。可以通过col.names参数改变默认的列名：

```
> records <- read.fwf("fixed-width.txt", widths=c(10,10,4,-1,4),
+                     col.names=c("Last","First","Born","Died"))
> records
        Last        First   Born   Died
1 Fisher     R.A.           1890   1962
```

```
2 Pearson    Karl        1857    1936
3 Cox        Gertrude    1900    1978
4 Yates      Frank       1902    1994
5 Smith      Kirstine    1878    1939
```

函数read.fwf默认以因子（分类变量）的形式读取非数值数据。例如上例中的“Last”列和“First”列都解释为因子。可以通过修改参数stringsAsFactors=FALSE使其还原为字符串型变量。

函数read.fwf还有许多其他阅读数据文件的巧妙实用的功能。read.fwf函数与read.table函数在许多方面类似。对于这两个函数的更多内容建议阅读帮助页面。

另请参阅

更多读取文本文件的讨论，参见方法4.7。

4.7 读取表格数据文件

问题

你希望读取一个包含表格数据的文本文件。

解决方案

使用read.table函数，结果会返回一个数据框：

```
> dfrm <- read.table("filename")
```

讨论

表格数据是很常见的，它们是有简单格式的文本文件。

- 每一行对应一条记录。
- 在每条记录中，不同数据域（或称为字段或变量）由一个分隔符隔开，比如空格、tab、冒号或者逗号。
- 每条记录包含相同数目的数据域（或称为字段或变量）。

这种形式的数据比起固定长度的格式更为灵活，因为每一栏的内容无需对齐排列。下面是方法4.6中数据的表格形式，其中使用空格作为分隔符：

```
Fisher R.A. 1890 1962
Pearson Karl 1857 1936
```

```
Cox Gertrude 1900 1978
Yates Frank 1902 1994
Smith Kirstine 1878 1939
```

函数read.table用于读取这类文件。默认情况下，它假定数据间以空格作为分隔符（空格或tab）：

```
> dfrm <- read.table("statisticians.txt")
> print(dfrm)
       V1       V2   V3   V4
1  Fisher     R.A. 1890 1962
2 Pearson     Karl 1857 1936
3     Cox Gertrude 1900 1978
4   Yates    Frank 1902 1994
5   Smith Kirstine 1878 1939
```

如果文件的分隔符不是空格，你可以修改sep参数来改变数据间的分隔符设置。

如果文件是用（:）作为字段分隔符，则可以用以下命令读取文件：

```
> dfrm <- read.table("statisticians.txt", sep=":")
```

尽管从该命令的打印结果中不会显示，但函数会把第一列和第二列解释为因子，而非字符串。我们可以通过以下命令查看其类型：

```
> class(dfrm$V1)
[1] "factor
```

为了防止read.table函数把字符串变量转化为因子，可以将stringsAsFactors参数设定为FALSE：

```
> dfrm <- read.table("statisticians.txt", stringsAsFactor=FALSE)
> class(dfrm$V1)
[1] "character"
```

此时第一列将为字符串，而非因子。

如果某一字段包含字符串“NA”，则read.table函数认为其是缺失值，并将其转换为NA。数据文件可能会以不同的符号标记缺失值，可以通过na.strings参数进行设置。例如，SAS习惯将缺失值标记为点(.)，我们可以通过以下方法读取这种数据：

```
> dfrm <- read.table("filename.txt", na.strings=".")
```

我是一位“自我解释型数据”的爱好者：文件中包含介绍其内容的信息。（计算机学家称文件包含了元数据。）read.table函数有两个特性支持阅读这类文件：第一，可以在数据文件的顶部添加标题栏，标题栏的内容对应每列变量的变量名，它使用与数据相同的分隔符来分隔标题。下面是文件添加了标题栏后得到的结果：

```
lastname firstname born died
Fisher R.A. 1890 1962
Pearson Karl 1857 1936
Cox Gertrude 1900 1978
Yates Frank 1902 1994
Smith Kirstine 1878 1939
```

现在我们可以告知read.table函数，数据文件中包含标题栏，函数会在生成数据框时自动为数据添加列名：

```
> dfrm <- read.table("statisticians.txt", header=TRUE, stringsAsFactor=FALSE)
> print(dfrm)
   lastname firstname born died
1    Fisher      R.A. 1890 1962
2   Pearson      Karl 1857 1936
3       Cox  Gertrude 1900 1978
4     Yates     Frank 1902 1994
5     Smith  Kirstine 1878 1939
```

read.table函数的第二个特性是注释行。以井号（#）开头的行内容都会被忽略，因此可以在这些行内添加注释内容：

```
# This is a data file of famous statisticians.
# Last edited on 1994-06-18
lastname firstname born died
Fisher R.A. 1890 1962
Pearson Karl 1857 1936
Cox Gertrude 1900 1978
Yates Frank 1902 1994
Smith Kirstine 1878 1939
```

read.table函数还有许多其他的参数，用于控制如何读取和解析数据。详情查看帮助页面。

另请参阅

如果你的数据文件是以逗号作为分隔符，参见方法4.8关于如何读取CSV文件。

4.8 读取CSV文件

问题

你希望读取逗号分隔值（Comma-Separated Value， CSV）文件中的数据。

解决方案

read.csv函数能够读取CSV格式的文件，如果CSV文件含有标题栏，可以使用以下命令：

```
> tbl <- read.csv("filename")
```

如果CSV文件不包含标题栏，则设定header选项为FALSE：

```
> tbl <- read.csv("filename", header=FALSE)
```

讨论

由于许多软件都使用CSV输入、输出数据，因此该格式普遍使用。例如R、Excel和其他电子表格软件、数据库软件，以及大多数统计软件包。CSV是一个表格形式的平面文件，文件的每一行是一行数据；每一行的数据项之间用逗号加以分隔。下面显示了一个简单的三行三列的CSV文件（第一行为标题栏，包含列名称，同样以逗号加以分隔）：

```
label,lbound,ubound
low,0,0.674
mid,0.674,1.64
high,1.64,2.33
```

函数read.csv读入数据并创建一个数据框，R一般以数据框格式表示表格数据。除非明确声明，否则函数read.csv假定文件有标题栏：

```
> tbl <- read.csv("table-data.csv")
> tbl
  label lbound ubound
1   low  0.000  0.674
2   mid  0.674  1.640
3  high  1.640  2.330
```

注意，read.csv函数会读取标题栏来作为数据框的列名称。

```
> tbl <- read.csv("table-data-with-no-header.csv", header=FALSE)
> tbl
    V1    V2    V3
1  low 0.000 0.674
2  mid 0.674 1.640
3 high 1.640 2.330
```

如果文件中不包含标题栏，则需要设定header参数为FALSE，R会自动生成一系列变量名称（此例中的V1、V2和V3）：

```
> str(tbl)
'data.frame': 3 obs. of 3 variables:
 $ label : Factor w/ 3 levels "high","low","mid": 2 3 1
 $ lbound: num 0 0.674 1.64
 $ ubound: num 0.674 1.64 2.33
```

Read.csv函数的一大优点在于它自动地将非数值型数据转换为因子（分类变量），对于

统计软件来说，这个结果一般是你希望得到的。数据框tbl中的label变量显示为因子而非字符串变量。可以通过命令查看tbl的数据结构：

```
> tbl <- read.csv("table-data.csv", as.is=TRUE)
> str(tbl)
'data.frame':   3 obs.  of  3 variables:
$ label : chr   "low" "mid" "high"
$ lbound: num   0 0.674 1.64
$ ubound: num   0.674 1.64 2.33
```

有时你希望数据解释为字符串而非因子。此时可以将as.is参数设为TRUE，即表示R不必将非数值型变量转换为因子：

```
> tbl <- read.csv("table-data.csv", as.is=TRUE)
> str(tbl)
'data.frame':   3 obs. of  3 variables:
 $ label : chr  "low" "mid" "high"
 $ lbound: num  0 0.674 1.64
 $ ubound: num  0.674 1.64 2.33
```

注意，变量label现在为字符型格式，而不再是因子。

另一个优点是以（#）符号开头的行在读取过程中会被忽略，这样就可以在数据文件中添加注释信息。可以通过comment.char=""设定取消这项特性。

函数read.csv有许多巧妙的功能。这些功能包括跳过输入文件中的第一行、控制某一列的转换、填充短行、限制每行的字符数，以及控制字符串是否用引号包括。详情查看R的帮助页面。

另请参阅

参见方法4.9。有关read.table函数的内容查看R帮助页面，函数read.table是函数read.csv的基础。

4.9 写入CSV文件

问题

你希望以逗号分隔符的格式保存矩阵或数据框内的数据。

解决方案

函数write.csv可以创建CSV文件：

```
> write.csv(x, file="filename", row.names=FALSE)
```

讨论

write.csv函数以CSV文件的格式将表格数据写到一个ASCII文件中。每行数据占用文件的一行，数据间以逗号（,）分隔：

```
> print(tbl)
  label lbound ubound
1   low  0.000  0.674
2   mid  0.674  1.640
3  high  1.640  2.330
> write.csv(tbl, file="table-data.csv", row.names=T)
```

上例在当前工作目录中生成了一个名为table-data.csv的文件，该文件内容如下：

```
"label","lbound","ubound"
"low",0,0.674
"mid",0.674,1.64
"high",1.64,2.33
```

注意，函数默认写入了数据的列名。可以通过参数col.names=FALSE改变该设定。

如果我们未指明row.names=FALSE，则函数会预先将数据的行名称属性（row.names）作为每一行的前置标记。如果数据不含有行名称，则函数write.csv会取行号作为每一行的前置标记，生成如下的CSV文件：

```
"","label","lbound","ubound"
"1","low",0,0.674
"2","mid",0.674,1.64
"3","high",1.64,2.33
```

我不怎么习惯在CSV文件中对每一行进行标记，因此我建议设定row.names=FALSE。

函数write.csv有意设为不甚灵活，因为它实在是想以CSV格式来写文件，因此你无法简单地改变其默认设置。可以使用write.table函数以其他格式保存表格数据。

write.csv函数的一大限制在于它无法向文件中追加行，可以使用write.table函数作为代替。

另请参阅

有关当前工作目录的内容，参见方法3.1；其他保存数据的方法参见方法4.14。

4.10 从网络中读取表格或CSV格式数据

问题

你希望直接把网络上的数据读入R工作空间。

解决方案

使用read.csv、read.table以及scan函数，并以URL地址替换原来的文件名。这些函数将直接读取远程服务器上的数据：

```
> tbl <- read.csv("http://www.example.com/download/data.csv")
```

你也可以通过URL链接来读取数据，这对于读取复杂数据而言是更好的方式。

讨论

网络中包含了大量的数据。你可以先下载这些数据到文件中，然后再使用R读取它们。但直接通过R读取网络数据会更为方便。在函数read.csv、read.table，或scan中设定URL地址(根据数据格式选择适当的函数)，数据将会下载并被解析。中间不会出差错。

除了使用URL以外，该方法大致类似于方法4.8中介绍的读取CSV格式文件和方法4.12介绍的读取复杂文件，因此这两个方法中的内容同样适用于本方法。

记住：URL对FTP服务器和HTTP服务器都有效。这意味着R也能使用URL读取FTP站点内的数据：

```
> tbl <- read.table("ftp://ftp.example.com/download/data.txt")
```

另请参阅

参见方法4.8和方法4.12。

4.11 读取HTML表格数据

问题

你希望读取网络中HTML网页的表格数据。

解决方案

使用R的XML包内的readHTMLTable函数，若需要读取页面内的所有表格数据，只要给出URL地址即可：

```
> library(XML)
> url <- 'http://www.example.com/data/table.html'
> tbls <- readHTMLTable(url)
```

若仅读取指定表格的数据，则需要设定which参数。下面的代码仅读取页面中第三个表格的数据：

```
> tbl <- readHTMLTable(url, which=3)
```

讨论

网页内可能包含多个HTML表格数据。使用readHTMLTable(url)函数能读取该页面中所有的表格并以列表的形式返回结果。这在浏览页面时很实用，但如果需要读取指定表格数据时则会变得很麻烦。此时需要设定参数which=*n*来选择需要的表格。根据你设定的参数，函数会读取第*n*个表格的数据。

下例为帮助页面中对readHTMLTable的演示，载入维基百科（Wikipedia）网页中标题为“World population”页面内的所有表格：

```
> library(XML)
> url <- 'http://en.wikipedia.org/wiki/World_population'
> tbls <- readHTMLTable(url)
```

结果显示该页面包含17个表格：

```
> length(tbls)
[1] 17
```

在下例中，我们仅需要这些表格中的第三个表格（该表格按国家列出它们的最多人口数量）。所以我们设定参数which=3：

```
> tbl <- readHTMLTable(url, which=3)
```

在下列表格中，第二列和第三列分别为国家名称和其人口数量：

```
> tbl[,c(2,3)]
                    Country / Territory    Population
1   Â People's Republic of China[44] 1,338,460,000
2                            Â India 1,182,800,000
3                    Â United States   309,659,000
4                        Â Indonesia   231,369,500
5                           Â Brazil   193,152,000
6                         Â Pakistan   169,928,500
7                       Â Bangladesh   162,221,000
8                          Â Nigeria   154,729,000
9                           Â Russia   141,927,297
10                           Â Japan   127,530,000
11                          Â Mexico   107,550,697
12                     Â Philippines    92,226,600
13                         Â Vietnam    85,789,573
14                         Â Germany    81,882,342
15                        Â Ethiopia    79,221,000
16                           Â Egypt    78,459,000
```

马上我们就观察到数据中有些问题：国家名称前端包含了一些奇怪的Unicode字符。其中的原因我也不很清楚，可能是由于维基百科页面中的格式所致。同时，“the People's Republic of China”（中华人民共和国）后面附带了“[44]”字符。在维基百科内，该符号是一个脚注引用，而此时则变为我们不需要的内容。更为尴尬的是，人口数量的数值中带有逗号，要转换成一般的数值会非常麻烦。所有这些问题都能通过某种字符操作来解决，但为了每个问题都需要在处理过程中添加至少一步或多步。

上面显示了读取HTML表格数据时遇到的主要问题。HTML网页主要是为了向人们展示数据，而不是向对计算机展示数据。网页中的数据对人来说很实用，但对计算机处理则比较麻烦。如果可以，尽量选择使用便于计算机读取的数据表现格式，例如XML、JSON和CSV 格式。

函数readHTMLTable存放于R的XML包中，而XML包又较为庞大且复杂。XML包依赖于一个名为libxml2的软件包，在使用前需要先下载并安装这一软件包。在Linux系统中，还需要安装R数据包所必需的Linux软件包xml2-config。

另请参阅

关于如何下载安装R包，例如XML包，参见方法3.9。

4.12 读取复杂格式数据文件

问题

你希望读取一个复杂或非常规格式的数据文件。

解决方案

- 使用readLines函数读取单独的行，然后以字符型变量的方式处理并提取数据项。
- 另外，使用scan函数单独读取每一个单元，并使用what参数描述文件中的单元流。该函数能将单元转换为数据，并对数据加以组合形成数据记录。

讨论

如果所有的数据都用同样整齐的表格形式存放，那么工作就会变得很轻松。我们就能使用read.table函数轻松完成任务。

然而实际中并非如此完美！

你总会遇到某种奇异的文件格式，而你的工作是无论如何都要将该文件导入R中。函数read.table和函数read.csv都是面向行来读取数据的，因此可能对这种数据无能为力。然而，readLines和scan函数能解决这类问题，因为它们能使你基于每一行甚至每一单元读取数据文件。

函数readLines比较简单。它读取文件中的各行并以字符串形式返回结果：

```
> lines <- readLines("input.txt")
```

可以通过设定参数n来限制读入的最大行数：

```
> lines <- readLines("input.txt", n=10)          # Read 10 lines and stop
```

scan函数内容更为丰富。它每次读取一个单元（token），并根据指令进行处理。其中第一个参数为文件名或者链接（之后将会介绍），第二个参数名为what，描述scan函数所期望的输入文件的单元。What参数的描述比较神秘但很奇妙：

what=numeric(0)
: 将下一单元译为数值。

what=integer(0)
: 将下一单元译为整数。

what=complex(0)
: 将下一单元译为复合型数值。

what=character(0)
: 将下一单元译为字符串。

what=logical(0)
: 将下一单元译为逻辑变量。

scan函数会重复执行指定的模式，直到所有数据被读取。

假设文件中含有一列简单的数据，如下所示：

```
2355.09 2246.73 1738.74 1841.01 2027.85
```

使用what=numeric(0)参数表达“该文档内包含一系列数值型的单元”：

```
> singles <- scan("singles.txt", what=numeric(0))
Read 5 items
> singles
[1] 2355.09 2246.73 1738.74 1841.01 2027.85
```

scan函数的一大特点在于，what参数可以是包含多种单元类型的列表。scan函数假定数据文件会重复这个单元类型列表。例如，假定文件中包含如下三元组数据形式：

```
15-Oct-87 2439.78 2345.63 16-Oct-87 2396.21 2,207.73
19-Oct-87 2164.16 1677.55 20-Oct-87 2067.47 1,616.21
21-Oct-87 2081.07 1951.76
```

以列表的形式预先告知scan函数，数据文件会出现三个单元重复出现的序列：

```
> triples <- scan("triples.txt", what=list(character(0),numeric(0),numeric(0)))
```

对列表中的元素添加名称，scan函数会将这些名称添加至对应的数据中：

```
> triples <- scan("triples.txt",
+                 what=list(date=character(0), high=numeric(0), low=numeric(0)))
Read 5 records
> triples
$date
[1] "15-Oct-87" "16-Oct-87" "19-Oct-87" "20-Oct-87" "21-Oct-87"
$high
[1] 2439.78 2396.21 2164.16 2067.47 2081.07
$low
[1] 2345.63 2207.73 1677.55 1616.21 1951.76
```

scan函数还有许多其他的使用方法，以下几个技巧尤为实用：

n=*number*

读取*number*个数据后停止读取（默认一直读取至文件末尾）。

nlines=*number*

读取*number*行数据后停止读取（默认一直读取至文件末尾）。

skip=*number*

读取数据前先跳过*number*行后再开始读取。

na.strings=*list*

将*list*中的字符串解释为NA。

示例

我们使用该方法从StatLib中读取数据，StatLib是由卡内基·梅隆大学建立和维护的统计软件与数据库。杰夫·维摩贡献了一个名为wseries的数据集（dataset），它展示了从1903年以来世界杯比赛输赢的情况。该数据集包含35行注释语句，之后是23行数据，存放于一个ASCII文件中。其中的数据部分如下所示：

```
1903  LWLlwwwW     1927  wwWW      1950  wwWW      1973  WLwllWW
1905  wLwWW        1928  WWww      1951  LWlwwW    1974  wlWWW
```

```
1906  wLwLwW     1929  wwLWW      1952  lwLWLww     1975  lwWLWlw
1907  WWww       1930  WWllwW     1953  WWllwW      1976  WWww
1908  wWLww      1931  LWwlwLW    1954  WWww        1977  WLwwlW

.
. (etc.)
.
```

该数据的编码如下：L表示在主场输球，l表示在客场输球，W表示在主场赢球，w表示在客场赢球。数据以列而非行的形式存储。这增加了处理的难度。

下面通过R软件读取这些数据：

```
# Read the wseries dataset:
#       - Skip the first 35 lines
#       - Then read 23 lines of data
#       - The data occurs in pairs: a year and a pattern (char string)
#
world.series <- scan("http://lib.stat.cmu.edu/datasets/wseries",
                     skip = 35,
                     nlines = 23,
                     what = list(year = integer(0),
                                 pattern = character(0)),
                     )
```

scan函数返回一个列表，其中包含两个元素：year和pattern。函数scan以从左到右的顺序读入数据，但由于数据以列的形式存储，所以显示结果的顺序较为奇怪：

```
> world.series$year
 [1] 1903 1927 1950 1973 1905 1928 1951 1974 1906 1929 1952
[12] 1975 1907 1930 1953 1976 1908 1931 1954 1977 1909 1932

.
. (etc.)
.
```

可以将列表根据年的顺序排列来解决这一问题：

```
> perm <- order(world.series$year)
> world.series <- list(year = world.series$year[perm],
+                      pattern = world.series$pattern[perm])
```

为了修正以上问题，我们按照年代顺序来对列表元素进行排序：

```
> world.series$year
 [1] 1903 1905 1906 1907 1908 1909 1910 1911 1912 1913 1914
[12] 1915 1916 1917 1918 1919 1920 1921 1922 1923 1924 1925

.
. (etc.)
.
```

```
> world.series$pattern
 [1] "LWLlwwwW" "wLwWW"    "wLwLwW"   "WWwww"    "wWLww"
 [6] "WLwlWlw"  "WWwlw"    "lWwWlW"   "wLwWlLW"  "wLwWw"
[11] "wwWW"     "lwWWw"    "WWlwW"    "WWllWw"   "wlwWLW"
[16] "WWlwwLLw" "wllWWWW"  "LlWwLwWw" "WWwW"     "LwLwWw"

.
. (etc.)
.
```

4.13 读取MySQL数据库中的数据

问题

你希望读取MySQL数据库中的数据。

解决方案

1. 安装RMySQL数据包。
2. 使用dbConnect函数打开一个数据库连接。
3. 使用dbGetQuery函数发起一个SELECT，并返回结果集。
4. 读取结束后，使用dbDisconnect函数终止与数据库的连接。

讨论

使用该方法前需要在计算机中安装RMySQL包。反过来，该R包需要MySQL客户端程序。若MySQL客户端程序未安装和配置，则需要参考MySQL的帮助文件，或向管理员寻求帮助。

使用dbConnect函数与MySQL数据库建立连接。它返回一个连接对象，之后的RMySQL函数的调用都需要用到这个连接：

```
library(RMySQL)
con <- dbConnect(MySQL(), user="userid", password="pswd",
                 host="hostname", client.flag=CLIENT_MULTI_RESULTS)
```

需要设定client.flag=CLIENT_MULTI_RESULTS来正确解决多结果集的情况。即使查询返回单一的结果集，你也必须按照上述要求设定client.flag参数，因为MySQL可能会在数据后附加多余的状态结果。

这里的用户名（username）、密码（password）和主机（host）等参数与通过*mysql*客户端程序访问MySQL的参数是一样的。上面例子中把这几个参数硬编码到dbConnect

函数调用中。实际中，这样的操作并不得当。因为这种操作方法将密码显示了出来，造成了信息安全问题。还有一个麻烦问题，即每当你修改密码和主机参数时，你必须再次找到该命令并进行修改。我强烈建议使用MySQL的安全机制作为替代。将三个参数添加至MySQL的配置文件中，其目录为*$HOME/.my.cnf*（UNIX下）和*C:\my.cnf*（Windows下），确保除你以外其他人无法访问该文件。文件内以标记符分隔内容，例如[client]。将参数填入[client]部分中，此时配置文件将包含如下内容：

```
[client]
user = userid
password = password
host = hostname
```

只要在配置文件中设定好这些参数，就不再需要在调用dbConnect时提供它们，这样便简化了操作过程：

```
con <- dbConnect(MySQL(), client.flag=CLIENT_MULTI_RESULTS)
```

使用dbGetQuery函数将你的SQL语句提交至数据库，并读取其结果集。完成该操作，首先需要有一个开放数据库链接：

```
sql <- "SELECT * from SurveyResults WHERE City = 'Chicago'"
rows <- dbGetQuery(con, sql)
```

当然你需要自行建立自己的SQL查询语句时，上述只是一个示例。你可以不局限于使用SELECT语句，任何能够生成结果集的SQL语句都可以使用。例如，我一般使用CALL命令，因为我的SQL语句封装在存储过程中，这些存储过程里包含了嵌入式的SELECT语句。

使用dbGetQuery很方便，因为它将结果集装入数据框中并返回该数据框。对SQL结果集来说，这是最好的显示方法。所以结果集会以行、列格式的表格数据显示，数据框也是。结果集中列的名称由SELECT语句确定，R把这些名称用于数据框的列名称。

在输出第一个结果集数据后，MySQL返回第二个包含状态信息的结果集。你可以选择检查状态信息或者忽略状态信息，但你必须阅读这些状态信息。否则，MySQL会显示当前有未处理的结果集，并停止运行。必要时可以调用dbNextResult函数：

```
if (dbMoreResults(con)) dbNextResult(con)
```

重复地调用dbGetQuery函数来完成多重查询，在每次调用后都检查结果的状态（需要时阅读其内容）。完成后，使用dbDisconnect断开数据库连接：

```
dbDisconnect(con)
```

下面是从股价数据库中读取并显示3行数据的全部过程。查询IBM股票在2008年最后3天的价格。假定用户名、密码以及主机已定义于*my.cnf*文件中：

```
> con <- dbConnect(MySQL(), client.flag=CLIENT_MULTI_RESULTS)
> sql <- paste("select * from DailyBar where Symbol = 'IBM'",
+              "and Day between '2008-12-29' and '2008-12-31'")
> rows <- dbGetQuery(con, sql)
> if (dbMoreResults(con)) dbNextResults(con)
> print(rows)
  Symbol        Day       Next OpenPx HighPx LowPx ClosePx AdjClosePx
1    IBM 2008-12-29 2008-12-30  81.72  81.72 79.68   81.25      81.25
2    IBM 2008-12-30 2008-12-31  81.83  83.64 81.52   83.55      83.55
3    IBM 2008-12-31 2009-01-02  83.50  85.00 83.50   84.16      84.16
  HistClosePx  Volume OpenInt
1       81.25 6062600      NA
2       83.55 5774400      NA
3       84.16 6667700      NA
> dbDisconnect(con)
[1] TRUE
```

另请参阅

有关RMySQL的配置和使用方法，参见方法3.9以及相应的帮助文件。

R还可以从其他关系数据库中读取数据，包括Oracle、Sybase、PostgreSQL以及SQLite。查看R基础发行版中自带的《R Data Import/Export》指导（参见方法1.6），同时该帮助文件也能通过CRAN网站（*http://cran.r-project.org/doc/manuals/R-data.pdf*）获得。

4.14 保存和传送目标

问题

你希望保存一个或多个R对象以便于以后使用，或者希望将该R对象复制至其他计算机中。

解决方案

使用save函数将对象写入一个文件中：

```
> save(myData, file="myData.RData")
```

在自己的计算机或其他支持R软件的平台中，使用load读取该文件的内容：

```
> load("myData.RData")
```

函数save以二进制数据方式保存数据，若需以ASCII码格式进行保存，则可以使用dput或dump函数：

```
> dput(myData, file="myData.txt")
> dump("myData", file="myData.txt")        # Note quotes around variable name
```

讨论

我一般将我的文件保存在工作空间中，但有时我也需要将文件保存在工作空间以外。比如，我有一个大型的数据对象，需要载入到其他的工作空间中，或者希望把R对象从Linux系统转存到Windows系统中。load和save函数能帮助我完成这一系列操作：save函数能将对象存储于一个跨平台的可携带的文件中，load函数能读取这类文件。

当使用load函数时，其本身不会返回数据，而是在工作空间中建立一些变量，将文件中的数据载入这些变量中，并（以列表方式）返回这些变量的名称。我第一次使用load函数时，我进行了这样的操作：

```
> myData <- load("myFile.RData")        # Achtung! Might not do what you think
```

我对结果感到十分疑惑，因为变量myData中不包含我的数据，而这些数据变量却已经存在于工作空间中。最终我阅读了load函数的帮助文件，了解了具体原因。

save函数以二进制的形式写入文件，以防止文件过大。有时你希望函数能以ASCII码格式输出。当你向邮件列表询问问题时，同时包含一段ASCII码数据代码可以允许别人能重现问题。这时，可以使用dput或dump函数，将结果以ASCII码格式输出。

对特殊R包中的目标进行保存和载入时尤其需要注意。当你载入该对象时，R不会自动载入该文件需要的R包，所以除非你事先已经载入了这些R包，否则系统将无法理解对象的内容。例如，假设你使用zoo包生成了一个名为z的目标文件，并且将该目标存放到名为*z.RData*的文件中，下面的命令会造成系统的误解：

```
> load("z.RData")       # Create and populate the z variable
> plot(z)               # Does not plot what we expected: zoo pkg not loaded
```

我们需要在显示或绘制zoo数据前，先载入zoo包：

```
> library(zoo)          # Load the zoo package into memory
> load("z.RData")       # Create and populate the z variable
> plot(z)               # Ahhh. Now plotting works correctly
```

第5章

数据结构

简介

第2章中讨论过，在R中运用向量（vector），便可以得到很多结果。本章超越对向量的探讨，讨论矩阵（matrices）、列表（list）、因子（factor）以及数据框（data frame）等方法。如果你学习过数据结构的相关概念，这里建议你先将它们搁置一边。因为R软件的数据结构将和其他语言的数据结构有所不同。

如果你想要学习技术层面的R软件数据结构，我建议你阅读《R in a Nutshell》（O'Reilly）和《R Language Definition》这两本书。本书的注解相对非正式。当我开始使用R软件时，以下这些是我所希望了解的概念。

向量

向量的主要性质包括：

向量是同质的

一个向量的所有元素（element）必须有相同的类型，或者用R的术语讲，有相同的模式（mode）。

向量可按照位置进行索引

因此`v[2]`指代向量v的第二个元素。

向量可按照多重位置索引，返回一个子向量

因此`v[c(2,3)]`是向量v的一个子向量（subvector），它包含v的第二个元素和第三个元素。

向量的元素可以被命名

向量有一个名称（names）属性，该属性的长度（length）等于向量自身的长度，名称属性赋给向量元素名称，例如：

```
> v <- c(10, 20, 30)
> names(v) <- c("Moe", "Larry", "Curly")
> print(v)
  Moe Larry Curly
   10    20    30
```

如果向量元素被命名，那么你能够用名称来选定它们

继续上面的例子：

```
> v["Larry"]
Larry
   20
```

列表

列表是非同质的

列表（list）可以包含不同形式的元素；用R软件术语来说，列表元素可能有不同的类型。列表甚至可以包含其他结构对象，例如列表和数据框；其允许创建循环的数据结构（data structure）。

列表可按位置索引

因此`lst[[2]]`指列表`lst`的第二个元素。注意这里的双方括号。

列表允许你从中抽取子列表

因此`lst[c(2,3)]`是列表`lst`的子列表，它包含原来列表`lst`的第二个元素和第三个元素。注意，这里用的是单方括号。

列表元素可以有名称

列表`lst[["Moe"]]`和列表`lst$Moe`都代表列表`lst`中名称为“Moe”的元素。

由于列表是非同质的，并且其元素能够按照名称检索，所以R中的列表与其他程序设计语言中的字典（dictionary）、散列表（hash），或查阅数据表（lookup table）一样（参见方法5.9）。令人惊奇的是，与大多数其他程序设计语言不同，在R软件中列表也可以按照位置索引。

模式：实体类型

在R软件中，每个对象都有一个模式，它表明该对象如何存储在存储器中：作为数值型（number）、字符串型（character string）、指向其他对象的指针列表（list of pointers），或者作为一个函数（function）等，例如：

对象	例子	模式
Number	`3.1415`	numeric
Vector of numbers	`c(2.7.182, 3.1415)`	numeric
Character string	`"Moe"`	character
Vector of character strings	`c("Moe", "Larry", "Curly")`	character
Factor	`factor(c("NY", "CA", "IL"))`	numeric
List	`list("Moe", "Larry", "Curly")`	list
Data frame	`data.frame(x=1:3, y=c("NY", "CA", "IL"))`	list
Function	`print`	function

R中的函数mode给出R对象的模式信息，例如：

```
> mode(3.1415)                        # Mode of a number
[1] "numeric"
> mode(c(2.7182, 3.1415))             # Mode of a vector of numbers
[1] "numeric"
> mode("Moe")                         # Mode of a character string
[1] "character"
> mode(list("Moe","Larry","Curly"))   # Mode of a list
[1] "list"
```

向量和列表的主要区别是：

- 在一个向量中，所有元素必须有相同的类型。
- 在一个列表中，元素可以有不同的类型。

类：抽象类型

在R软件中，每一个对象（object）同样有一个定义它们抽象类型的类（class）。这个术语从面向对象的程序设计中借用而来。单个数字可以代表许多不同含义：一段距离、一个时间点或者重量。由于所有这些对象作为一个数字存储，它们都有一个“数值型”类型；然而它们可以用不同的类来表明其含义。

例如，一个日期（Date）对象是由单个数值构成：

```
> d <- as.Date("2010-03-15")
> mode(d)
[1] "numeric"
> length(d)
[1] 1
```

它有一个日期类（Date）来告诉我们怎样解释这个数值：即，自1970年1月1日以来的天数。

```
> class(d)
[1] "Date"
```

在R软件中，对象所在的类将决定如何处理这个对象。例如，泛型函数print有专用版本（称为方法）来打印归属于不同类（class）的对象，例如属于类data.frame，Date，lm的对象有不同的打印处理。当你打印一个对象，R软件根据对象的类调用相应的print函数。

纯量

纯量的奇特之处在于它与向量的关系。在某些软件中，向量和纯量是两个截然不同的概念。而在R软件中，它们是相同的：纯量是仅包含唯一一个元素的向量。本书时常使用术语“纯量”，但其仅是“拥有唯一元素的向量”的简写。

考虑R中的常量pi，它是一个纯量：

```
> pi
[1] 3.141593
```

鉴于纯量是单元素的向量，所以可以对pi用向量函数：

```
> length(pi)
[1] 1
```

你可以索引它。当然它有且仅有的元素为π：

```
> pi[1]
[1] 3.141593
```

如果要寻找第二个元素，那么你将一无所获：

```
> pi[2]
[1] NA
```

矩阵

在R软件中，矩阵（matrices）只是有维数（dimension）的向量。它也许乍看有些生疏，但你可以把维数赋予一个向量将其简单地转换成一个矩阵。

向量有一个属性称为维数（dim），其初始值为NULL，如下所示。

```
> A <- 1:6
> dim(A)
```

```
NULL
> print(A)
[1] 1 2 3 4 5 6
```

当设置一个向量的dim属性时，便将维数赋予了它。现在我们将向量维数设置成2 × 3并输出它，看看会发生什么：

```
> dim(A) <- c(2,3)
> print(A)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

瞧！这个向量被重塑成了一个2 × 3的矩阵。

矩阵也可以由列表来创建。与向量一样，列表也有一个dim属性，其初始值为空值（即NULL）：

```
> B <- list(1,2,3,4,5,6)
> dim(B)
NULL
```

如果我们设置dim属性，它将给列表一个形式：

```
> dim(B) <- c(2,3)
> print(B)
     [,1] [,2] [,3]
[1,] 1    3    5
[2,] 2    4    6
```

瞧！我们将列表转变成了一个2 × 3的矩阵。

数组（array）

对矩阵的讨论可以推广到3维或者*n*维结构。只要将更多的维数指定给相关向量（或列表）就可以了。接下来的例子创建了一个维数为2 × 3 × 2的3维数组：

```
> D <- 1:12
> dim(D) <- c(2,3,2)
> print(D)
, , 1

     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
```

注意，由于不可能在二维介质中输出3维结构，R软件一次仅输出结构的一个“片段”。

由列表生成矩阵

让我感觉非常奇怪的是，仅仅将dim属性赋予一个列表，就能将该列表转换成矩阵。这变得更加奇怪了。

考虑到列表元素可以是异质的（混合类型）。我们可以从一个异质的列表开始，赋予它维数，从而创建一个异质的矩阵。以下代码片段创建了一个由数值和字符数据组成的矩阵：

```
> C <- list(1, 2, 3, "X", "Y", "Z")
> dim(C) <- c(2,3)
> print(C)
     [,1] [,2] [,3]
[1,] 1    3    "Y"
[2,] 2    "X"  "Z"
```

对我来说这显得有些奇怪，因为我通常假定一个矩阵由纯粹数值组成，而非数值和字母的混合。R软件没有这种限制。

出现一个异质矩阵的可能性看似有种强大而不可思议的魅力。然而，当你用矩阵做标准的矩阵运算时，它会产生问题。例如，当上述的矩阵C用于矩阵乘法时会发生什么呢？如果将它转换成数据框又会发生什么呢？这时候，会得到很奇怪的答案。

本书通常忽视异质矩阵这一“病态”情况。假设你应用的是一个简单的、纯数值的矩阵。如果矩阵包含混合数据，有些有关矩阵的方法可能会产生奇怪的效果（或者根本没有结果输出）。例如，将这样的混合矩阵转换成向量或者数据框时会出现问题（参见方法5.33）。

因子

因子（factor）看起来和向量相似，但它拥有特殊的性质。R软件记录向量中的唯一值，每一个唯一值称为相关联因子的水平（level）。R软件对因子使用精简的表示方法，使其能在数据框中有效存储。在其他程序设计语言中，因子会由一个枚举值的向量来表示。

以下是因子的两个关键应用：

分类变量

一个因子可以代表一个分类变量。分类变量用于列联表（contingency table）、线性回归（linear regression）、方差分析（ANOVA）、逻辑回归（logistic regression）以及许多其他领域。

分组

这是一个根据数据组别来分类或标记数据项的技术。详见第6章的“简介”。

数据框

数据框（data frame）是一个强大而灵活的结构。大多数正式的R应用软件包含数据框。数据框旨在模拟数据集，与SAS或SPSS中的数据集一样。

数据框是一种表格式的（或矩形的）数据结构，它有行和列。但是，数据框不是由矩阵来实现的，一个数据框实际上是一个列表：

- 列表的元素是向量或者因子[注1]。
- 这些向量或者因子构成数据框的列。
- 所有向量或者因子必须有相同的长度，换句话说，所有列必须有相同的高度（height）。
- 这些相等高度的列使数据框呈矩形形状。
- 数据框的列必须命名。

由于数据框既是列表又是矩形结构，所以R软件提供了两种不同的模式来访问数据框的内容：

- 可以使用列表操作符来提取数据框的列，例如`dfrm[i]`、`dfrm[[i]]`或者`dfrm$`*`name`*。
- 可以使用类似矩阵中的记号，例如`dfrm[i,j]`、`dfrm[i,]`或者`dfrm[,j]`。

你对数据框的理解很可能取决于你的知识背景：

统计人员

数据框是一张观测值的表格。每一行包含一个观测。每个观测必须包含相同的变量。这些变量称为列，并且你可以通过名称访问它们。你也可以通过行号和列号来查阅观测的内容，就像对待矩阵那样。

结构化查询语言（SQL）的程序员

数据框是一张表格。表格完全驻留在内存中，你可以将它保存为一个文本文件，然后可以恢复它为数据框。你无需说明列类型，因为R可以自动辨识出列类型。

注1：　一个数据框可以由向量、因子以及矩阵混合构成。矩阵的列成为数据框的列。每个矩阵中的行数必须与向量和因子的长度匹配。换言之，数据框中的所有元素必须有相同的高度。

Excel使用者

数据框是一张工作表，或者说是一定范围内的工作表。它更有限制性，然而工作表中的每一列有一个类型。

SAS使用者

数据框是一个SAS数据集，其中的所有数据存储于内存中。R软件可以将数据读、写进磁盘，但是在R软件处理它时，数据框必须处于存储器中。

R程序员

数据框是一个混合数据结构，它的一部分为矩阵，另一部分为列表。数据框的一列可以包含数字、字符串或者因子但不能是它们的混合。你可以像索引一个矩阵那样检索数据框。数据框也是一张列表，其中列表的元素为数据框的列，因此你可以使用列表操作符来访问它们。

计算机科学家

数据框是一个矩形数据结构。每列都是强类型，并且每列必须是数值型、字符串或者因子。列必须有标签，行可以有标签或者没有标签。该表格可以根据位置检索，或者根据列名或行名检索。它也可以被列表操作符访问，在这种情况下，R将数据框看做一个列表，其元素为数据框的列。

主管

你可以将姓名和数字输入数据框。这很简单！一个数据框像一个小型数据库。你的职员将会很乐意使用数据框。

5.1 对向量添加数据

问题

你需要对向量添加额外的数据项。

解决方案

使用向量构造函数（c）来构造含有额外数据项的向量，例如：

```
> v <- c(v,newItems)
```

对于单一数据项，你也可以把添加的新数据项赋给下一个向量元素。R软件会自动扩充向量：

```
> v[length(v)+1] <- newItem
```

讨论

如果询问我如何附加一个数据项给一个向量，我会建议你也许不必这么做。

警告： 当你考虑整个向量而非单个数据项时，R软件会最大程度地发挥作用。你是否不断地附加数据项给一个向量？如果是这样的话，那么你可能在一个循环中工作。对于小的向量，这不成问题；但对于大的向量，程序会运行缓慢。当你反复用一个元素扩展向量时，R软件的内存管理将会有问题。试着用向量水平操作替代循环，你将会写更少的代码，并且R软件会更快地运行。

然而，人们偶尔需要附加数据给向量。我的经验表明，它可以通过使用向量构造函数（c）先构造一个新向量，然后把含有新、旧数据的向量合并为一个，这是最有效率的方法。对于附加单个元素或者多个元素，这个方法都十分奏效：

```
> v <- c(1,2,3)
> v <- c(v,4)          # Append a single value to v
> v
[1] 1 2 3 4
> w <- c(5,6,7,8)
> v <- c(v,w)          # Append an entire vector to v
> v
[1] 1 2 3 4 5 6 7 8
```

你也可以通过将一个数据项指定到超过向量结尾的位置来附加它，如同解决方案中所示。事实上，R软件对于扩展向量十分自由。你可以赋值给任何元素，而R会扩展向量来满足你的需求：

```
> v <- c(1,2,3)        # Create a vector of three elements
> v[10] <- 10          # Assign to the 10th element
> v                    # R extends the vector automatically
 [1] 1 2 3 NA NA NA NA NA NA 10
```

注意，R软件对下标越界并不报错。它只是用缺失值（NA）来填充以扩展向量至需要的长度。

R软件有一个append函数，它通过将数据项附加给一个已有的向量来构造新的向量。然而，我的经验表明这个函数的运行速度比向量构造函数和元素赋值都要慢。

5.2 在向量中插入数据

问题

你需要将一个或者多个数据值插入一个向量中。

解决方案

不考虑其名称，函数append通过使用after参数在一个向量中插入数据，该参数给出新数据项的插入点：

```
> append(vec, newvalues, after=n)
```

讨论

新数据项的插入位置由after参数给出。以下例子是在一个数列中间（第5个元素之后）插入数值99：

```
> append(1:10, 99, after=5)
 [1]  1  2  3  4  5 99  6  7  8  9 10
```

特殊值after=0，意味着在向量的最前面插入新数据项：

```
> append(1:10, 99, after=0)
 [1] 99  1  2  3  4  5  6  7  8  9 10
```

方法5.1的注解在这里也适用。如果你将单个数据项插入一个向量，你也许考虑元素水平。然而在向量水平工作将会使你更容易地编码，并且软件也会运行得更快。

5.3 理解循环规则

问题

你想要理解奇妙的循环规则，它指导R软件如何处理不等长度的向量。

讨论

当做向量运算时，R软件执行元素对元素的操作。当两个向量有相同长度时，这个操作很奏效：R软件成对搭配这些向量的元素并且将操作应用于这些元素对。

但是，当两个向量的长度不相等时会发生什么呢？

在这种情况下，R软件应用循环规则。R从两个向量的第一个元素开始，成对处理向量元素。在某个元素位置，较短的向量已经处理完所有的元素，而较长的向量仍然有未处理的元素。这时候，R软件返回较短向量的开始位置，“循环”应用它的元素；同时继续在较长向量中处理元素；直到这个操作完成为止。它根据需要来循环较短向量的元素，直到操作完成。

可以进行如下的循环规则的可视化。下面是两个向量的图表，1：6的整数和1：3的整数，如下图所示。

1:6	1:3
1	1
2	2
3	3
4	
5	
6	

显然，向量1:6比向量1:3长。如果用(1:6) + (1:3)进行向量的加法，看上去向量1:3的元素太少了。然而，R软件循环应用向量1:3的元素，像下图这样成对搭配两个向量并且产生一个有 6 个元素的向量：

1:6	1:3	(1:6) + (1:3)
1	1	2
2	2	4
3	3	6
4	*1*	5
5	*2*	7
6	*3*	9

以下是你在命令行所看见的：

```
> (1 : 6) + (1 : 3)
[1] 2 4 6 5 7 9
```

不仅向量操作能调用循环规则，函数也可以。函数cbind可以用来构造列向量，比如下面的列向量1:6和1:3。当然这两列有不同的高度：

```
> cbind(1 : 6)
     [,1]
[1,]    1
[2,]    2
[3,]    3
[4,]    4
[5,]    5
[6,]    6

> cbind(1 : 3)
     [,1]
```

```
     [1,]    1
     [2,]    2
     [3,]    3
```

如果我们尝试将这两个列连接起来变成一个有两列的矩阵，那么它们的长度是不匹配的。由于向量1：3太短，所以函数cbind调用循环规则并且循环应用它的元素：

```
> cbind(1 : 6, 1 : 3)
     [,1] [,2]
[1,]    1    1
[2,]    2    2
[3,]    3    3
[4,]    4    1
[5,]    5    2
[6,]    6    3
```

如果较长向量的长度不是较短向量长度的倍数，R将给出一个警告。由于这种操作很值得怀疑，并且运算逻辑中可能有漏洞，所以给出警告是个好现象：

```
> (1 : 6) + (1 : 5)          # Oops! 1:5 is one element too short
[1] 2 4 6 8 10 7
Warning message:
In (1 : 6) + (1 : 5) :
    longer object length is not a multiple of shorter object length
```

一旦你理解了循环规则，你就会意识到在向量和纯量间的运算仅仅是这个规则的应用。在下面例子中，10被循环应用直到向量加法完成：

```
> (1 : 6) + 10
[1] 11 12 13 14 15 16
```

5.4 构建因子（即分类变量）

问题

你有一个字符串向量或整数向量。你需要R将它们作为一个因子，R的术语将其叫做分类变量（categorical variable）。

解决方案

factor函数把离散值向量编码成一个因子：

```
> f <- factor(v)          # v is a vector of strings or integers
```

如果向量仅包含一些可能值的子集而并非全集，那么可以应用第二个参数来给出因子所有可能的水平（level）：

```
> f <- factor(v, levels)
```

讨论

在R软件中，每一个分类变量的可能值称为一个水平（level）。一个由水平值构成的向量叫做因子（factor）。因子非常符合R软件面向向量的特色，并且它们在R处理数据和建立统计模型方面有广泛的应用。

大多数情况下，可以通过调用factor函数将分类数据转换成一个因子，该函数会识别分类数据的不同水平并且将它们打包成一个因子：

```
> f <- factor(c("Win","Win","Lose","Tie","Win","Lose"))
> f
[1] Win  Win  Lose Tie  Win  Lose
Levels: Lose Tie Win
```

注意，当我们输出因子f时，R软件没有对它的输出值使用引号。这里的输出值是水平而非字符串。还要注意，当我们输出因子时，R软件也在因子下方显示了其不同的水平。

如果向量仅包含所有可能水平的一个子集，那么R软件会输出所有可能水平的不完整视图。假设你有一个字符串值的变量，记为wday，它表示数据观测的日期位于一周中的哪一天。例如，

```
> f <- factor(wday)
> f
 [1] Wed Thu Mon Wed Thu Thu Thu Tue Thu Tue
Levels: Mon Thu Tue Wed
```

R认为周一、周四、周二和周三是仅有的可能水平。周五没有列出。显然，实验室人员从来没有在周五做观测，所以R不知道周五是一个可能值。因此你需要明确地列出wday的因子的所有可能水平：

```
> f <- factor(wday, c("Mon","Tue","Wed","Thu","Fri"))
> f
 [1] Wed Thu Mon Wed Thu Thu Thu Tue Thu Tue
Levels: Mon Tue Wed Thu Fri
```

现在R知道f是一个有五个可能水平的因子。它同样知道它们的正确顺序。R软件本来将周四放在周二之前，因为它错误地假定使用字母顺序[注1]。第二个参数明确定义了正确的顺序。

在许多情况下，你并不需要显式地调用函数factor。当R函数需要因子时，它通常自动将数据转换成因子。例如，table函数仅用于因子变量，所以自动地将它的输入转换成因子。当需要指定全部水平取值或者当需要控制水平的排序时，必须显式地创建一个因子变量。

注1： 更确切地说，它根据计算机系统的Local设置对名称进行排序。

另请参阅

参见方法12.6从连续数据创建一个因子。

5.5 将多个向量合并成单个向量以及平行因子

问题

有几组数据，每组包含一个向量。需要将这些向量合并成一个长向量并且同时创建一个平行因子来识别每个值的原始组。

解决方案

创建一个列表来包含这些向量。运用stack函数将列表合并为一个两列数据框：

```
> comb <- stack(list(v1=v1, v2=v2, v3=v3))     # Combine 3 vectors
```

该数据框的两个列分别称为values和ind。第一列包含数据，第二列包含它的平行因子。

讨论

为何要将所有的数据合并成一个长向量和一个平行因子呢？这是因为许多重要的统计函数需要这种格式的数据。

假设你调查大一、大二以及大三学生的自信心水平（比如“你在学校里感到自信的时间百分比是多少？”）。现在有三个向量，分别是freshmen、sophomores和juniors。你需要对组间差异进行方差分析。方差分析函数aov需要一个包含调查结果的向量和一个识别这些组的平行因子。可以用stack函数合并这些组：

```
> comb <- stack(list(fresh=freshmen, soph=sophomores, jrs=juniors))
> print(comb)
   values   ind
1    0.60 fresh
2    0.35 fresh
3    0.44 fresh
4    0.62 fresh
5    0.60 fresh
6    0.70  soph
7    0.61  soph
8    0.63  soph
9    0.87  soph
10   0.85  soph
11   0.70  soph
12   0.64  soph
13   0.76   jrs
14   0.71   jrs
```

```
15    0.92   jrs
16    0.87   jrs
```

现在可以对这两列进行方差分析：

```
> aov(values ~ ind, data=comb)
```

当构建这个列表时，我们必须对列表元素提供标签（在这个例子中，标签为fresh、soph和jrs）。由于stack函数将标签用做平行因子的水平，所以它们是必不可少的。

5.6 创建列表

问题

需要创建并填充一个列表。

解决方案

用函数list来从单个数据项创建列表：

```
> lst <- list(x, y, z)
```

讨论

列表可以十分简单，例如下面包含三个数的列表：

```
> lst <- list(0.5, 0.841, 0.977)
> lst
[[1]]
[1] 0.5

[[2]]
[1] 0.841

[[3]]
[1] 0.977
```

当R打印列表时，它根据每个列表元素的位置（[[1]]、[[2]]、[[3]]）来识别它们并且在它们位置的下方打印元素值（例如，[1] 0.5）。

更有用的是，列表并不像向量，它可以包含不同模式（类型）的元素。下面是一个极端的例子，它是由一个纯量、字符串、向量和函数混合组成的列表：

```
> lst <- list(3.14, "Moe", c(1,1,2,3), mean)
> lst
[[1]]
[1] 3.14
```

```
[[2]]
[1] "Moe"

[[3]]
[1] 1 1 2 3

[[4]]
function (x, ...)
UseMethod("mean")
<environment: namespace:base>
```

也可以通过先创建一个空列表，然后填充该列表的方法来构造列表。下面是用这个方法构造的混合列表：

```
> lst <- list()
> lst[[1]] <- 3.14
> lst[[2]] <- "Moe"
> lst[[3]] <- c(1,1,2,3)
> lst[[4]] <- mean
```

列表元素可以命名。函数list允许你为每一个元素提供名称：

```
> lst <- list(mid=0.5, right=0.841, far.right=0.977)
> lst
$mid
[1] 0.5

$right
[1] 0.841

$far.right
[1] 0.977
```

另请参阅

参见本章“引言”中更多有关列表的内容；参见方法5.9用命名元素构建和应用列表。

5.7 根据位置选定列表元素

问题

需要根据位置访问列表元素。

解决方案

有以下方法可供选择。其中，lst是一个列表变量：

`lst[[n]]`

　　从列表中选定第n个元素。

`lst[c(n1, n2, ..., nk)]`

　　返回一个元素的列表，所选元素由相应的位置值确定。

注意，第一个例子将返回单个元素而第二个例子将返回一个列表。

讨论

假设我们有一个包含四个整数的列表，称为years：

```
> years <- list(1960, 1964, 1976, 1994)
> years
[[1]]
[1] 1960

[[2]]
[1] 1964

[[3]]
[1] 1976

[[4]]
[1] 1994
```

我们可以用上述的双方括号的语法来访问列表的单个元素：

```
> years[[1]]
[1] 1960
```

我们也可以用单方括号的语法来提取子列表（sublists）：

```
> years[c(1,2)]
[[1]]
[1] 1960

[[2]]
[1] 1964
```

上述两个例子的语法有微妙的差别，容易造成混淆：`lst[[n]]`和`lst[n]`之间有重要的区别。它们不是等价的：

`lst[[n]]`

　　返回一个元素，而非列表。它是lst的第n个元素。

`lst[n]`

　　返回一个列表，而非元素。这个列表包含一个元素，它取自lst的第n个元素。它是

lst[c(n_1,n_2,…,n_k)]的一个特例，在此我们没有用c(...)这样的结构，因为这里只有一个数值：n。

当我们查看上述结果的结构时，它们的差异将变得很明显——一个是数字，另一个则是列表：

```
> class(years[[1]])
[1] "numeric"

> class(years[1])
[1] "list"
```

当我们用函数cat输出它们的值时，明显的差异会变得十分恼人。函数cat可以打印基本数据值，但是无法打印结构化对象：

```
> cat(years[[1]], "\n")
1960
> cat(years[1], "\n")
Error in cat(list(...), file, sep, fill, labels, append) :
  argument 1 (type 'list') cannot be handled by 'cat'
```

很幸运R对这个问题做出了警告。在其他情况下，可能会经历漫长而艰辛的努力后发现你访问了一个子表而并非你想要的某个元素，反之亦然。

5.8 根据名称选定列表元素

问题

需要根据名称访问一个列表的元素。

解决方案

有以下方法可供选定。下面的例子中是一个列表：

lst[["*name*"]]

选定名称为*name*的元素。若该名称的元素在列表中不存在，则返回NULL。

lst[["*name*"]]

和上面的例子等价，只是语法不同。

lst[c($name_1$, $name_2$, ..., $name_k$)]

返回一个由函数lst中的参数所决定的元素而构成的列表。

注意，前两个例子返回一个元素而第三个例子返回一个列表。

讨论

列表中的每个元素都可以有一个名称。如果列表元素有名称，则可以根据元素的名称来选定元素。下面的例子创建了一个包含四个命名整数的列表：

```
> years <- list(Kennedy=1960, Johnson=1964, Carter=1976, Clinton=1994)
```

以下两个例子返回相同的值，即命名为“Kennedy”的元素：

```
> years[["Kennedy"]]
[1] 1960
> years$Kennedy
[1] 1960
```

以下两个例子返回从years中提取的子表：

```
> years[c("Kennedy","Johnson")]
$Kennedy
[1] 1960

$Johnson
[1] 1964

> years["Carter"]
$Carter
[1] 1976
```

前面提到根据位置选定列表元素的两个例子（即单括号和双括号，具体参见方法5.7）。这里lst[["*name*"]]和lst["*name*"]之间也类似的区别，它们是完全不同的：

lst[["*name*"]]

: 返回一个元素，并非列表。

lst["*name*"]

: 返回一个列表，而不是元素，它是lst[c[$name_1$, $name_2$, ..., $name_k$]]的一个特例。这里由于只有一个名字（name），所以没有采用c(...)的形式。

另请参阅

有关根据位置而非名称访问元素的方法，参见方法5.7。

5.9 构建一个名称/值关联表

问题

需要创建一个联合名称和值的列表——就像其他程序设计语言中的字典（dictionary）、散列表（hash），或者查找表（lookup table）。

解决方案

函数list允许将名称赋予元素，创建一个名称和值的关联。

```
> lst <- list(mid=0.5, right=0.841, far.right=0.977)
```

如果有名称和值的平行向量，可以创建一个空列表并用向量化赋值语句填充它：

```
> lst <- list()
> lst[names] <- values
```

讨论

列表中的每一个元素都能够命名，而且可以根据名称检索列表元素。这给了你一个基本的程序设计工具：关联名称和值的能力。

当创建列表时，可以同时给出元素的名称。函数list允许形如***name=value***的参数：

```
> lst <- list(
+             far.left=0.023,
+             left=0.159,
+             mid=0.500,
+             right=0.841,
+             far.right=0.977)
>             lst
$far.left
[1] 0.023

$left
[1] 0.159

$mid
[1] 0.5

$right
[1] 0.841

$far.right
[1] 0.977
```

命名元素的一个方法是创建一个空列表并通过赋值语句填充它：

```
> lst <- list()
> lst$far.left <- 0.023
> lst$left <- 0.159
> lst$mid <- 0.500
> lst$right <- 0.841
> lst$far.right <- 0.977
```

有时有一个名称向量和一个与相应值有关的向量：

```
> values <- pnorm(-2:2)
> names <- c("far.left", "left", "mid", "right", "far.right")
```

可以创建一个空列表，然后用向量化赋值语句填充它，该方法可以关联名称和名称对应的值：

```
> lst <- list()
> lst[names] <- values
> lst
$far.left
[1] 0.02275013

$left
[1] 0.1586553

$mid
[1] 0.5

$right
[1] 0.8413447

$far.right
[1] 0.9772499
```

一旦关联建立，列表可以通过一个简单的列表查找将名称“翻译”成它的值：

```
> cat("The left limit is", lst[["left"]], "\n")
The left limit is 0.1586553
> cat("The right limit is", lst[["right"]], "\n")
The right limit is 0.8413447

> for (nm in names(lst)) cat("The", nm, "limit is", lst[[nm]], "\n")
The far.left limit is 0.02275013
The left limit is 0.1586553
The mid limit is 0.5
The right limit is 0.8413447
The far.right limit is 0.9772499
```

5.10 从列表中移除元素

问题

需要从列表中移除某个元素。

解决方案

将NULL赋给要移除的元素，R会将该元素从列表中移除。

讨论

要移除一个列表元素，可以根据位置或名称选定它，然后将NULL赋值给选定的元素：

```
> years
$Kennedy
[1] 1960

$Johnson
[1] 1964

$Carter
[1] 1976

$Clinton
[1] 1994

> years[["Johnson"]] <- NULL          # Remove the element labeled "Johnson"
> years
$Kennedy
[1] 1960

$Carter
[1] 1976

$Clinton
[1] 1994
```

可以用这个方法移除多个元素：

```
> years[c("Carter","Clinton")] <- NULL          # Remove two elements
> years
$Kennedy
[1] 1960
```

5.11 将列表转换为向量

问题

需要把一个列表转换为一个向量（即把列表中的所有元素“展平”赋值给一个向量）。

解决方案

运用函数unlist。

讨论

许多情况下我们需要一个向量。R中的基本统计函数需要一个向量参数而非列表。例如，如果iq.scores是一个包含数值的列表，那么我们不能直接计算它们的平均值：

```
> mean(iq.scores)
[1] NA
Warning message:
In mean.default(iq.scores) :
    argument is not numeric or logical: returning NA
```

替代的方法是，我们必须用函数unlist将列表转换成向量，然后计算向量的平均值：

```
> mean(unlist(iq.scores))
[1] 106.4452
```

下面是另一个例子。我们可以用函数cat处理纯量和向量，但我们不能用它处理列表：

```
> cat(iq.scores, "\n")
Error in cat(list(...), file, sep, fill, labels, append) :
argument 1 (type 'list') cannot be handled by 'cat'
```

一个解决方案是在打印前把列表转化为向量：

```
> cat("IQ Scores:", unlist(iq.scores), "\n")
IQ Scores: 89.73383 116.5565 113.0454
```

另请参阅

这里所讨论的转换方法在5.33中有更详细的解释。

5.12 从列表中移除取值为空值（即NULL）的元素

问题

列表包含NULL。需要移除它们。

解决方案

假设lst是一个部分元素为NULL的列表。以下表达式将会移除NULL元素：

```
> lst[sapply(lst, is.null)] <- NULL
```

讨论

从列表中找到并移除NULL元素会是惊人地微妙。我在尝试了其他一些方法后写下了下列表达式，其中包括了一些明显的表达式以及缺点。下面介绍R如何工作：

1. R调用sapply来使函数is.null应用于列表的每一个元素。
2. sapply返回逻辑值向量。如果相应的列表元素是NULL，返回TRUE值。
3. R根据步骤2中的逻辑向量从列表中选定相应取值为TRUE的列表元素。
4. R把NULL赋值给选定的元素，将它们从列表中移除。

考虑到我们通过设置元素值为NULL来移除它们（参见方法5.10），好奇的读者可能会疑惑列表怎么会包含NULL元素。答案是我们可以创建一个包含NULL元素的列表：

```
> lst <- list("Moe", NULL, "Curly")     # Create list with NULL element
> lst
[[1]]
[1] "Moe"

[[2]]
NULL

[[3]]
[1] "Curly"

> lst[sapply(lst, is.null)] <- NULL     # Remove NULL element from list
> lst
[[1]]
[1] "Moe"

[[2]]
[1] "Curly"
```

另请参阅

关于如何移除列表元素，请参见方法5.10。

5.13 使用条件来移除列表元素

问题

需要按照条件检验从列表中移除元素，例如移除负值或者小于某些临界值的元素。

解决方案

基于条件构造一个逻辑向量。使用该逻辑向量选定列表元素并将NULL赋值给这些元素。例如，以下赋值语句旨在从列表中移除所有负值：

```
> lst[lst < 0] <- NULL
```

讨论

这个方法基于R的两个有用的特性。首先，可以通过一个逻辑向量索引列表。无论什么情况下只要向量元素为TRUE，它相应的列表元素就可以选定。其次，可以通过把NULL赋值给一个列表元素的方法来移除它。

假设需要把取值为0的元素从列表lst中移除。构建一个可以识别这些不需要值的逻辑向量（lst == 0）。随后从列表中选定这些元素并且将NULL值赋给它们：

```
> lst[lst == 0] <- NULL
```

下列表达式将从列表中移除值为NA的元素：

```
> lst[is.na(lst)] <- NULL
```

到目前为止，一切都很好。但当不能够轻易地构建逻辑向量时，问题就出现了。这种情况经常出现在当需要使用函数，但它不能处理列表时。例如，假设需要移除绝对值小于1的列表元素，但不幸的是，函数abs不能处理列表：

```
> lst[abs(lst) < 1] <- NULL
Error in abs(lst) : non-numeric argument to function
```

最简单的解决方案是调用函数unlist把列表转换为向量，然后检验它：

```
> lst[abs(unlist(lst)) < 1] <- NULL
```

更巧妙的解决方案是使用lapply把函数应用于列表中的每一个元素：

```
> lst[lapply(lst,abs) < 1] <- NULL
```

列表也可以保留复合对象，而不只是原子值。假设mods是一个由函数lm创建的线性模型列表。以下表达式将移除任何R^2值小于0.30的模型：

```
> mods[sapply(mods, function(m) summary(m)$r.squared < 0.3)] <- NULL
```

另请参阅

参见方法5.7、方法5.10、方法5.11、方法6.2和方法11.1。

5.14 矩阵初始化

问题

需要创建一个矩阵并且用给定值对它进行初始化。

解决方案

从一个向量或者列表中获得数据，然后运用函数matrix使数据形成一个矩阵。下面的例子使一个向量形成了一个2 × 3的矩阵（即两行三列）：

```
> matrix(vec, 2, 3)
```

讨论

假设我们需要创建并初始化一个2 × 3的矩阵，我们可以从一个向量中获得初始数据，然后用函数matrix来把它转化为矩阵：

```
> theData <- c(1.1, 1.2, 2.1, 2.2, 3.1, 3.2)
> mat <- matrix(theData, 2, 3)
> mat
     [,1]     [,2]     [,3]
[1,] 1.1      2.1      3.1
[2,] 1.2      2.2      3.2
```

matrix的第一个参数是数据，第二个参数是行数，第三个参数是列数。注意，矩阵是逐列而非逐行填充的。

我们通常用一个值，例如0或者NA来初始化整个矩阵。如果matrix的第一个参数是单一值，那么R运用循环规则自动复制这个值来填充整个矩阵：

```
> matrix(0, 2, 3)              # Create an all-zeros matrix
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
> matrix(NA, 2, 3)             # Create a matrix populated with NA
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
```

可以用单行创建一个矩阵，当然，它会很难阅读：

```
> mat <- matrix(c(1.1, 1.2, 1.3, 2.1, 2.2, 2.3), 2, 3)
```

R的一个通用习惯是以矩形形状键入数据来显示矩阵结构：

```
> theData <- c(1.1, 1.2, 1.3,
+              2.1, 2.2, 2.3)
> mat <- matrix(theData, 2, 3, byrow=TRUE)
```

设置byrow=TRUE告诉matrix数据是逐行而非逐列的（这是默认值）。也可以通过一个命令来完成，例如：

```
> mat <- matrix(c(1.1, 1.2, 1.3,
+                 2.1, 2.2, 2.3),
+               2, 3, byrow=TRUE)
```

用这种方式表示，读者可以很快看到这个两行三列的数据。

一个把向量变成矩阵的应急方法是：把维数赋值给向量。这部分在本章“简介”中有所讨论。下面的例子创建了一个纯数值的向量并把它转化成一个2 × 3的矩阵：

```
> v <- c(1.1, 1.2, 1.3, 2.1, 2.2, 2.3)
> dim(v) <- c(2,3)
> v
     [,1]  [,2]  [,3]
[1,]  1.1   1.3   2.2
[2,]  1.2   2.1   2.3
```

对我而言，我发现这比使用matrix更加不透明，尤其因为这里没有byrow选项。

另请参阅

参见方法5.3。

5.15 执行矩阵运算

问题

需要执行矩阵运算，例如矩阵转置、矩阵求逆、矩阵乘法或者构造一个单位矩阵。

解决方案

`t(A)`

　　矩阵A的转置。

`solve(A)`

　　矩阵A求逆。

`A %*% B`

　　矩阵A与矩阵B相乘。

diag(n)

一个*n*阶对角（单位）矩阵。

讨论

A*B是逐个元素的乘法运算，然而A%*%B是矩阵乘法。

所有这些函数返回一个矩阵。它们的参数可以是矩阵或者数据框。如果是数据框，那么R首先将它们转化成矩阵（虽然只有当数据框只包含数值型数据时才有意义）。

5.16 将描述性名称赋给矩阵的行和列

问题

需要把描述性名称赋值给一个矩阵的行或列。

解决方案

每个矩阵都有一个rownames属性和一个colnames属性。下面例子把字符串向量赋值给矩阵的相应属性：

```
> rownames(mat) <- c("rowname1", "rowname2", ..., "rownamem")
> colnames(mat) <- c("colname1", "colname2", ..., "colnamen")
```

讨论

R允许你把名称赋值给一个矩阵的行和列，这对输出这个矩阵很有用。如果名称定义了，R会显示它们，增强输出的可读性。考虑下例中有关IBM、Microsoft和Google公司股票价格的相关系数矩阵：

```
> print(tech.corr)
      [,1]  [,2]  [,3]
[1,] 1.000 0.556 0.390
[2,] 0.556 1.000 0.444
[3,] 0.390 0.444 1.000
```

在这个形式中，矩阵输出的结果很难解释。如果我们对该矩阵的行和列定义名称，那么R会用这些名称来注释矩阵输出：

```
> colnames(tech.corr) <- c("IBM","MSFT","GOOG")
> rownames(tech.corr) <- c("IBM","MSFT","GOOG")
> print(tech.corr)
      IBM  MSFT  GOOG
IBM 1.000 0.556 0.390
```

```
MSFT  0.556  1.000   0.444
GOOG  0.390  0.444   1.000
```

现在读者一眼便能看出哪些行和列与哪些股票有关。

命名行和列的另一个优势是，可以通过矩阵元素的名称来查阅它们：

```
> tech.corr["IBM","GOOG"]      # What is the correlation between IBM and GOOG?
[1] 0.39
```

5.17 从矩阵中选定一行或一列

问题

需要从矩阵中选定一行或者一列。

解决方案

解决方案取决于输出结果的形式。如果结果是一个简单的向量，就可以使用普通索引：

```
> vec <- mat[1,]        # First row
> vec <- mat[,3]        # Third column
```

如果结果是一个单行矩阵或者一个单列矩阵，就要包含参数drop=FALSE：

```
> row <- mat[1,,drop=FALSE]              # First row in a one-row matrix
> col <- mat[,3,drop=FALSE]              # Third column in a one-column matrix
```

讨论

通常，当从矩阵中选定一行或者一列时，R会移除矩阵的维数属性，得到的结果是一个没有维数属性的向量：

```
> mat[1,]
[1] 1 4 7 10
> mat[,3]
[1] 7 8 9
```

然而，当包含参数drop=FALSE，R会保留维数属性。在这种情况下，选定一行返回单行向量（一个$1 \times n$的矩阵）：

```
> mat[1,,drop=FALSE]
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
```

同样，用参数drop=FALSE选定一列返回单列向量（一个$n \times 1$的矩阵）：

```
> mat[,3,drop=FALSE]
     [,1]
[1,]    7
[2,]    8
[3,]    9
```

5.18 用列数据初始化数据框

问题

数据按列来组织，需要将它们组合成一个数据框。

解决方案

如果数据由多个向量和（或者）因子来表示，用函数data.frame将它们组合成一个数据框：

```
> dfrm <- data.frame(v1, v2, v3, f1, f2)
```

如果数据由一个包含向量和（或者）因子的列表来表示，用函数as.data.frame：

```
> dfrm <- as.data.frame(list.of.vectors)
```

讨论

数据框是列的集合，数据框的每一个列相当于一个观测变量（这里是就统计意义而言，非编程意义）。如果数据已经组织成列的形式，那么构建一个数据框是很容易的。

函数data.frame能够用向量构建一个数据框，其中每个向量是一个观测变量。假设有两个数值型的预测变量，它们是一个分类预测变量和一个因变量。函数data.frame可以用它们来创建一个数据框：

```
> dfrm <- data.frame(pred1, pred2, pred3, resp)
> dfrm
        pred1       pred2 pred3     resp
1  -2.7528917 -1.40784130    AM 12.57715
2  -0.3626909  0.31286963    AM 21.02418
3  -1.0416039 -0.69685664    PM 18.94694
4   1.2666820 -1.27511434    PM 18.98153
5   0.7806372 -0.27292745    AM 19.59455
6  -1.0832624  0.73383339    AM 20.71605
7  -2.0883305  0.96816822    PM 22.70062
8  -0.7063653 -0.84476203    PM 18.40691
9  -0.8394022  0.31530793    PM 21.00930
10 -0.4966884 -0.08030948    AM 19.31253
```

注意，函数data.frame从程序变量中得到列名。可以显式地提供列名来覆盖默认的列名：

```
> dfrm <- data.frame(p1=pred1, p2=pred2, p3=pred3, r=resp)
> dfrm
          p1          p2 p3        r
1 -2.7528917 -1.40784130 AM 12.57715
2 -0.3626909  0.31286963 AM 21.02418
3 -1.0416039 -0.69685664 PM 18.94694
.
. (etc.)
.
```

数据也许组织成向量，但这些向量是在一个列表里，并非个别程序变量，如下所示。

```
> lst <- list(p1=pred1, p2=pred2, p3=pred3, r=resp)
```

这时可以使用函数as.data.frame把向量列表转换为一个数据框：

```
> as.data.frame(lst)
          p1          p2 p3        r
1 -2.7528917 -1.40784130 AM 12.57715
2 -0.3626909  0.31286963 AM 21.02418
3 -1.0416039 -0.69685664 PM 18.94694
.
. (etc.)
.
```

5.19 由行数据初始化数据框

问题

数据已组织成行，而你需要将它们组合成一个数据框。

解决方案

将每一行数据存储在一个单行数据框中。再将这个单行数据框存储在一个列表中。调用函数rbind和do.call把多行结合成一个大数据框。

```
> dfrm <- do.call(rbind, obs)
```

这里，obs是一个单行数据框列表。

讨论

数据往往以观测值集合的形式出现。每个观测（observation）是一个包含多个值的记录，其中的每个值代表一个观测变量。一个平面文件的行通常是这样的：每一行是一个

记录，每个记录包含多列，每一列代表一个不同的变量（参见方法4.12）。这种数据以观测（observation）格式而不是以变量格式来组织。换句话说，你每次得到的是一行而不是一列。

每个这样的行可能会以多种方式存储。一个显而易见的方式是把它存储为向量。如果有纯粹的数值型数据（numerical data），就可以用向量存储。

然而，许多数据集是数值、字符和分类数据的混合，在这种情况下，不能用向量进行存储。我推荐把每个异质的行存储在一个单行数据框内。（可以把每一行存储在一个列表中，但这种方法会稍稍复杂一些。）

举个具体例子，假设有10行数据，其中每个观测有4个变量：pred1、pred2、pred3和resp。每一行存储在一个单行数据框中，所以有10个这样的数据框。这些数据框存储在一个叫做obs的列表中。obs的第一个元素看起来如下所示。

```
> obs[[1]]
   pred1 pred2 pred3   resp
1 -1.197  0.36    AM 18.701
```

如果观测值存储在向量中而非单行数据框中，这个方法也适用。

我们需要将这些行组合成一个数据框。这时可以应用函数rbind来完成。它把输入参数结合在一起，使得每一个参数成为结果中的一行。例如，如果我们用rbind组合前两个观测值，我们会得到一个有两行的数据框，即

```
> rbind(obs[[1]], obs[[2]])
   pred1 pred2 pred3   resp
1 -1.197  0.36    AM 18.701
2 -0.952  1.23    PM 25.709
```

我们需要把每个观测值结合在一起，而不只是前两个。所以我们应用R的向量处理功能。函数do.call会将obs扩展为一个长参数列表，并用它来调用函数rbind：

```
> do.call(rbind,obs)
    pred1  pred2 pred3   resp
1  -1.197  0.360    AM 18.701
2  -0.952  1.230    PM 25.709
3   0.279  0.423    PM 21.572
4  -1.445 -1.846    AM 14.392
5   0.822 -0.246    AM 19.841
6   1.247  1.254    PM 25.637
7  -0.394  1.563    AM 24.585
8  -1.248 -1.264    PM 16.770
9  -0.652 -2.344    PM 14.915
10 -1.171 -0.776    PM 17.948
```

这个结果是由数据行构建的一个数据框。

有时，由于在你的控制范围之外，数据行存储在列表中而非单行数据框中。例如，也许你需要处理一个由数据库包返回的行。在这种情况下，obs是一个列表的列表，而不是一个数据框列表。

我们首先调用Map函数将行数据转换成数据框数据，然后应用这个方法：

```
> dfrm <- do.call(rbind,Map(as.data.frame,obs))
```

另请参阅

如果数据组织成列，而非行，请参见方法5.18；参见方法12.18学习更多关于函数do.call的内容。

5.20 添加行至数据框

问题

你需要给数据框添加一行或更多新行。

解决方案

创建第二个包含这些新行的临时数据框。然后调用函数rbind将这个临时数据框添加到原始数据框后面。

讨论

假设我们有一个数据框是关于芝加哥大都会区的城市，现在需要添加一个新行至该数据框中。首先，我们用新数据创建一个单行数据框：

```
> newRow <- data.frame(city="West Dundee", county="Kane", state="IL", pop=5428)
```

然后，我们调用函数rbind添加这个单行数据框至我们现有的数据框：

```
> suburbs <- rbind(suburbs, newRow)
> suburbs
             city    county  state     pop
1         Chicago      Cook     IL 2853114
2         Kenosha   Kenosha     WI   90352
3          Aurora      Kane     IL  171782
4           Elgin      Kane     IL   94487
5            Gary  Lake(IN)     IN  102746
6          Joliet   Kendall     IL  106221
7      Naperville    DuPage     IL  147779
8 ArlingtonHeights     Cook     IL   76031
9     Bolingbrook      Will     IL   70834
```

```
10          Cicero      Cook   IL  72616
11        Evanston      Cook   IL  74239
12         Hammond  Lake(IN)   IN  83048
13        Palatine      Cook   IL  67232
14      Schaumburg      Cook   IL  75386
15          Skokie      Cook   IL  63348
16        Waukegan  Lake(IL)   IL  91452
17     West Dundee      Kane   IL   5428
```

函数rbind告诉R我们正在添加一个新行至suburbs，而不是一个新列。对你而言，显而易见的是newRow为一行而非一列，但对R并非如此。（调用函数cbind添加一列。）

值得注意的是，新行必须使用与原来数据框相同的列名；否则，rbind将无法运行。

当然，我们可以将这两个步骤合并为一步：

```
> suburbs <- rbind(suburbs,
+                  data.frame(city="West Dundee", county="Kane", state="IL", pop=5428))
```

由于rbind允许多个参数，所以我们甚至可以把这种技巧推广至添加多个新行。

```
> suburbs <- rbind(suburbs,
+                  data.frame(city="West Dundee", county="Kane", state="IL", pop=5428),
+                  data.frame(city="East Dundee", county="Kane", state="IL", pop=2955))
```

警告： 不要使用这个方法给大数据框添加许多行。这会促使R反复地重新分配一个大的数据结构，该过程将会十分缓慢。此时，需要运用更加有效的方法构建数据框，例如方法5.19或方法5.21中所示。

5.21 预分配数据框

问题

当你逐行构建数据框时，你需要预先分配空间而不是递增累加地添加行。

解决方案

由通用向量和因子来创建数据框，即通过调用函数numeric(*n*)、character(*n*)和factor(*n*)，例如：

```
> dfrm <- data.frame(colname1=numeric(n), colname2=character(n), ... etc. ... )
```

这里，*n*是数据框需要的行数。

讨论

理论上讲，你可以通过逐个添加新行来构建一个数据框。对于小数据框，这是可行的，但用这种方法构建一个大数据框就太繁琐了。当重复地添加新行至一个大的数据结构时，R的内存管理器会不良运转。因此R代码运行会极为缓慢。

预先分配数据框的一个解决方案是——假设你知道必需的行数。通过一次性地预先分配数据框，就回避了内存管理器的问题。

假设你需要创建一个有1 000 000行、3列的数据框，它有两个数值型列和一个字符型列。调用函数numeric和character预先分配列；然后调用data.frame把它们连接在一起：

```
> N <- 1000000
> dfrm <- data.frame(dosage=numeric(N), lab=character(N), response=numeric(N))
```

现在你有了一个维数正确的数据框，1 000 000 × 3，它等待着接收内容。

数据框可以包含因子，但预先分配一个因子需要一点技巧。你不能简单地调用函数factor(*n*)。由于你正在创建一个因子，所以你需要指定它的水平。继续前面的例子，假设你需要把字符串类型的lab列转换为一个因子类型的列，并且可能的水平是NJ、IL和CA。可以在列的声明中包含水平，如下所示。

```
> N <- 1000000
> dfrm <- data.frame(dosage=numeric(N),
+                    lab=factor(N, levels=c("NJ", "IL", "CA")),
+                    response=numeric(N) )
```

5.22 根据位置选择数据框的列

问题

你需要根据位置从数据框中选择列。

解决方案

对于选定的单个列，可以应用列表运算符，例如：

dfrm[[*n*]]

返回一列——具体来说，返回数据框dfrm的第*n*列。

对于选定一列或者更多列，并且将它们打包到一个数据框内，你可以调用下面的子列表表达式：

dfrm[*n*]

返回一个数据框，该数据框的组成元素仅为dfrm的第*n*列。

dfrm[c(n_1, n_2, ..., n_k)]

返回一个数据框，它由数据框dfrm的第$n_1, n_2, \cdots, n_k$列构建而成。

你可以用矩阵型下标来选定一列或更多列：

dfrm[, *n*]

返回第*n*列（假设*n*是一个单值）。

dfrm[, c(n_1, n_2, ..., n_k)]

返回一个数据框，它由位置为$n_1, n_2, \cdots, n_k$的列构建而成。

注意，矩阵型下标可以返回两个不同的数据类型（列或者数据框），取决于你是选定一列或者多列。

讨论

从数据框中选定列有许多令人困惑的方法。除非你理解它们背后的逻辑，否则这些选项会让人十分迷惑。当你阅读这个说明时，特别注意语法中的微小变化是怎样改变表达式的意思的，例如一个逗号、一个双括号。

让我们以芝加哥大都会区16个最大的城市人口数据为例：

```
> suburbs
             city   county state     pop
1         Chicago     Cook    IL 2853114
2         Kenosha  Kenosha    WI   90352
3          Aurora     Kane    IL  171782
4           Elgin     Kane    IL   94487
5            Gary Lake(IN)    IN  102746
6          Joliet  Kendall    IL  106221
7      Naperville   DuPage    IL  147779
8 Arlington Heights   Cook    IL   76031
9     Bolingbrook     Will    IL   70834
10         Cicero     Cook    IL   72616
11       Evanston     Cook    IL   74239
12        Hammond Lake(IN)    IN   83048
13       Palatine     Cook    IL   67232
14     Schaumburg     Cook    IL   75386
15         Skokie     Cook    IL   63348
16       Waukegan Lake(IL)    IL   91452
```

使用简单的列表符号来精确地选定一列，例如第一列：

```
> suburbs[[1]]
[1] "Chicago"           "Kenosha"           "Aurora"            "Elgin"
```

```
[5] "Gary"          "Joliet"        "Naperville"    "Arlington Heights"
[9] "Bolingbrook"   "Cicero"        "Evanston"      "Hammond"
[13] "Palatine"     "Schaumburg"    "Skokie"        "Waukegan"
```

suburbs的第一列是一个向量，所以suburbs[[1]]也返回了一个向量。如果第一列是一个因子，我们便会得到一个因子。

当使用单括号时，结果会有所不同，例如suburbs[1]或者suburbs[c(1,3)]。你依然得到要求的列，但R把它们转换成了一个数据框。下面的例子返回数据框中的第一列：

```
> suburbs[1]
                city
1            Chicago
2            Kenosha
3             Aurora
4              Elgin
5               Gary
6             Joliet
7         Naperville
8  Arlington Heights
9        Bolingbrook
10            Cicero
11          Evanston
12           Hammond
13          Palatine
14        Schaumburg
15            Skokie
16          Waukegan
```

下一个例子返回数据框中的第一列和第三列：

```
> suburbs[c(1,3)]
                city     pop
1            Chicago 2853114
2            Kenosha   90352
3             Aurora  171782
4              Elgin   94487
5               Gary  102746
6             Joliet  106221
7         Naperville  147779
8  Arlington Heights   76031
9        Bolingbrook   70834
10            Cicero   72616
11          Evanston   74239
12           Hammond   83048
13          Palatine   67232
14        Schaumburg   75386
15            Skokie   63348
16          Waukegan   91452
```

混淆的一个主要来源在于suburbs[[1]]和suburbs[1]，它们看起来相似但会产生截然不同的结果：

`suburbs[[1]]`

　　它返回一列。

`suburbs[1]`

　　它返回一个数据框，并且这个数据框仅包含一列。这是dfrm[c(n_1, n_2, ..., n_k)]的一个特例。我们不需要c(...)结构，因为这里只有一个n。

这里的重点是“一列”与“包含一列的数据框”有所不同。第一个表达式返回一列，所以它是一个向量或者一个因子。第二个表达式返回一个数据框，这是截然不同的结果。

如同“解决方案”中所示，R允许你使用矩阵符号来选定列。这里的“可能会得到不同结果”可能会令你疑惑：你可能得到一列或者一个数据框，这取决于你用的下标。如果是一个简单下标，你将得到一列，如下所示。

```
> suburbs[,1]
 [1] "Chicago"          "Kenosha"          "Aurora"           "Elgin"
 [5] "Gary"             "Joliet"           "Naperville"       "Arlington Heights"
 [9] "Bolingbrook"      "Cicero"           "Evanston"         "Hammond"
[13] "Palatine"         "Schaumburg"       "Skokie"           "Waukegan"
```

但运用相同类型的矩阵型语法，如果是多个下标（即向量下标），R将返回一个数据框：

```
> suburbs[,c(1,4)]
                city     pop
1            Chicago 2853114
2            Kenosha   90352
3             Aurora  171782
4              Elgin   94487
5               Gary  102746
6             Joliet  106221
7         Naperville  147779
8  Arlington Heights   76031
9        Bolingbrook   70834
10            Cicero   72616
11          Evanston   74239
12           Hammond   83048
13          Palatine   67232
14        Schaumburg   75386
15            Skokie   63348
16          Waukegan   91452
```

这里产生了一个问题。假设你在一些老的R脚本中看到以下表达式：

```
dfrm[,vec]
```

请马上回答，它返回一列还是一个数据框呢？这要依情况而定。如果vec包含一个值，那么得到的是一列；否则，得到一个数据框。你不能仅仅从语法来辨别。

为了避免这个问题，你可以在下标中包含drop=FALSE，它促使R返回一个数据框：

```
dfrm[,vec,drop=FALSE]
```

现在，关于这个返回的数据结构没有歧义了。它是一个数据框。

总而言之，运用矩阵符号从数据框中选定列不是最好的方法。我推荐你应该使用先前描述的列表运算符。它们似乎更加清楚。

另请参阅

参见方法5.17更多关于使用drop=FALSE的内容。

5.23 根据列名选定数据框的列

问题

你需要根据列名从一个数据框中选定相应的列。

解决方案

对于选定单一列，你可以使用下列这些列表表达式之一：

`dfrm[["name"]]`
: 返回一列，列名为*name*。

`dfrm$name`
: 与前面相同，只是使用了不同的语法。

选定一列或者更多列并且将它们转换成一个数据框，可以使用下列列表表达式：

`dfrm["name"]`
: 选定一列并且把它转换成一个数据框对象。

`dfrm[c("name_1", "name_2", ..., "name_k")]`
: 选定多列并且把它们转换成一个数据框。

你可以使用矩阵型下标来选定一列或者更多列：

`dfrm[, "name"]`
: 返回一个命名的列。

`dfrm[, c("name_1", "name_2", ..., "name_k")]`
: 选定多列并且将它们打包转换成一个数据框。

同样，矩阵型下标可以返回两个不同的数据类型（列或者数据框），这取决于你选定一列还是多列。

讨论

数据框中的所有列必须有名称。如果你知道它们的名称，根据名称而非位置来选定它们会更加方便和易读。

刚才描述的解决方案与方法5.22中的解决方案相似，在方法5.22中我们根据位置选定列。唯一的不同在于本节中我们使用列名而非列号。方法5.22中的所有注意事项也适用于本节：

- `dfrm[["name"]]`返回一列，而不是一个数据框。
- `dfrm[c("name1", "name2", ..., "namek")]`返回一个数据框，而不是一列。
- `dfrm["name"]`是先前表达式的一个特例，所以它返回一个数据框，而不是一列。
- 矩阵型下标可以返回一列或者一个数据框，所以要注意你提供名称的个数。关于这个细节和使用`drop=FALSE`的讨论，请参见方法5.22。

以下是一个新的补充：

```
dfrm$name
```

它实际上与`dfrm[["name"]]`相同，但它更加易于输入和阅读。

另请参阅

参见方法5.22来理解这些选定列的方法。

5.24 更便捷地选定行和列

问题

你需要一个更容易的方法来从数据框或者矩阵中选定行和列。

解决方案

调用函数`subset`。参数`select`是需要选定的列名，或者列名向量：

```
> subset(dfrm, select=colname)
> subset(dfrm, select=c(colname1, ..., colnamek))
```

注意，列名不需要加引号。

参数subset是选定行的一个逻辑表达式。在这个表达式中，可以把列名作为逻辑表达式的一部分。本例中，response是数据框的一列，我们选择response为正值的行：

```
> subset(dfrm, subset=(response > 0))
```

当把参数select和subset合并时，subset会起到最大作用：

```
> subset(dfrm, select=c(predictor,response), subset=(response > 0))
```

讨论

正如在方法5.22和方法5.23中描述的那样，建立索引（indexing）是从数据框中选定行和列的"正式"途径。然而，当索引表达式变得复杂时，建立索引会是冗长难处理的。

函数subset提供了一个更加方便和可读的方法来选定行和列。它的奇妙之处在于可以直接在用于选定行和列的表达式中引用数据框的这些列。

下面的例子应用了MASS软件包中的Cars93数据集。前面讲过数据集包含列Manufacturer、Model、MPG.city、MPG.highway、Min.Price以及Max.Price：

选定在城市中每加仑汽油行驶超过30英里的汽车的型号名称

```
> subset(Cars93, select=Model, subset=(MPG.city > 30))
     Model
31 Festiva
39   Metro
42   Civic
.
. (etc.)
.
```

选定美国制造的四缸汽车的型号名称和价格区间

```
> subset(Cars93, select=c(Model,Min.Price,Max.Price),
+        subset=(Cylinders == 4 & Origin == "USA"))
        Model  Min.Price Max.Price
6     Century       14.2      17.3
12   Cavalier        8.5      18.3
13    Corsica       11.4      11.4
.
. (etc.)
.
```

选定所有在高速公路上每加仑汽油行驶的英里数的值高于中位数的汽车的制造商名和型号名

```
> subset(Cars93, select=c(Manufacturer,Model),
+        subset=c(MPG.highway > median(MPG.highway)))
   Manufacturer      Model
1         Acura    Integra
5           BMW       535i
6         Buick    Century
.
. (etc.)
.
```

实际上，函数subset的功能远大于本节描述的内容。它也可以从列表和向量中选定对象。详情参见帮助说明页。

5.25 修改数据框的列名

问题

把矩阵或列表转变为数据框。R将名称赋予列，但这个名称毫无意义并且十分荒诞。

解决方案

数据框包含一个colnames属性，它是一个由列名构成的向量。你可以更新单个名称或者整个向量，例如：

```
> colnames(dfrm) <- newnames              # newnames is a vector of character strings
```

讨论

数据框的列必须有名称。如果你将一个纯数值的（vanilla）矩阵转变成一个数据框，R将会给出合理但枯燥的名称——例如，V1、V2、V3等：

```
> mat
       [,1]   [,2]   [,3]
[1,] -0.818 -0.667 -0.494
[2,] -0.819 -0.946 -0.205
[3,]  0.385  1.531 -0.611
[4,] -2.155 -0.535 -0.316
> as.data.frame(mat)
      V1     V2     V3
1 -0.818 -0.667 -0.494
2 -0.819 -0.946  0.205
3  0.385  1.531 -0.611
4 -2.155 -0.535 -0.316
```

如果矩阵包含已定义的列名，R会使用这些名称而不是合成新的名称。

然而，将列表转换为数据框会产生一些奇怪的合成名称：

```
> lst
[[1]]
[1] -0.284 -1.114 -1.097 -0.873

[[2]]
[1] -1.673 0.929 0.306 0.778

[[3]]
[1] 0.323 0.368 0.067 -0.080

> as.data.frame(lst)
  c..0.284...1.114...1.097...0.873. c..1.673..0.929..0.306..0.778.
1                            -0.284                         -1.673
2                            -1.114                          0.929
3                            -1.097                          0.306
4                            -0.873                          0.778
  c.0.323..0.368..0.067...0.08.
1                         0.323
2                         0.368
3                         0.067
4                        -0.080
```

同样，如果列元素已有名称，R就会使用它们。

幸运的是，你可以通过设置colnames属性用你自己定义的名称覆盖合成的列名称：

```
> dfrm <- as.data.frame(lst)
> colnames(dfrm) <- c("before","treatment","after")
> dfrm
  before treatment  after
1 -0.284    -1.673  0.323
2 -1.114     0.929  0.368
3 -1.097     0.306  0.067
4 -0.873     0.778 -0.080
```

另请参阅

参见方法5.33。

5.26 编辑数据框

问题

数据框中有错误或者遗失的数据。你需要用一个便捷的方法编辑数据框的内容。

解决方案

R包含一个数据编辑器，它可以在电子数据表窗口中显示你的数据框。使用函数edit调用这个编辑器：

```
> temp <- edit(dfrm)
> dfrm <- temp                    # Overwrite only if you're happy with the changes!
```

你可以修改你想要的内容，然后关闭这个编辑器窗口。函数edit返回更新后的数据，这里将它指派给了temp。如果你满意这次的修改，就可用该结果覆盖你的数据框。

如果你对修改有很大信心，则可以调用fix函数来启动编辑器，它会自动用修改后的结果覆盖你的变量。然而，这里没有办法"撤销"以返回原始数据。

```
> fix(dfrm)
```

讨论

图5-1是在Windows操作系统中的R数据编辑器的截图，它是编辑方法5.22中数据内容的截图。

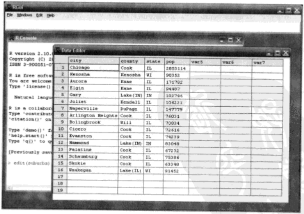


图5-1：编辑数据框

在我写本书时，编辑器还相当原始。它不包含现代编辑器的共同特点——例如，它甚至没有"撤销"命令。我不推荐在日常使用中调用数据编辑器，但它可用于紧急情况下的修饰。

由于没有撤销，注意这个解决方案将编辑结果指派给一个临时的、起中间作用的变量。如果你觉得你的数据被搞乱了，你就可以删除这个临时的变量而不会影响原始数据。

另请参阅

多种R添加包的图形用户界面（GUI）优于R提供的简单的数据编辑器。

5.27 从数据框中移除NA值

问题

你的数据框包含NA值，它给你带来了麻烦。

解决方案

调用na.omit移除包含NA值的行。

```
> clean <- na.omit(dfrm)
```

讨论

我经常发现这样的情况，数据框中的一些NA值使得所有数据看起来四分五裂。一个解决方案是调用na.omit简单地移除所有包含NA的行。

下面我们可以看到由于输入中包含NA值，函数cumsum运算失败，即

```
> dfrm
           x          y
1 -0.9714511 -0.4578746
2         NA  3.1663282
3  0.3367627         NA
4  1.7520504  0.7406335
5  0.4918786  1.4543427
> cumsum(dfrm)
          x          y
1 -0.971451 -0.4578746
2        NA  2.7084536
3        NA         NA
4        NA         NA
5        NA         NA
```

如果我们移除NA值，函数cumsum可以完成它的累加求和（summations）：

```
> cumsum(na.omit(dfrm))
           x          y
1 -0.9714511 -0.4578746
4  0.7805993  0.2827589
5  1.2724779  1.7371016
```

这个方法对向量和矩阵也奏效，但对列表不起作用。

你仍有足够的数据吗

这里，一个显而易见的威胁是简单地从数据中除去含有缺失数据的记录会使结果在计算机或者统计意义上毫无价值。在分析中要确保删除缺失数据是有意义的。记住na.omit会移除整行，不只是NA值，它会删除许多有用的信息。

5.28 根据名称排除列

问题

你需要利用列名把该列从数据框中排除它。

解决方案

调用包含一个select参数值为负值的subset函数：

```
> subset(dfrm, select = -badboy)        # All columns except badboy
```

讨论

我们可以根据位置排除一列（例如，dfrm[-1]），但我们要怎样根据名称排除一列呢？subset函数可以从一个数据框中排除列。参数select通常是需要选择的列的一个列表，但在这个名称前加上一个减号标志（-）就使这些列被排除了。

当计算一个数据框的相关系数矩阵时，我经常遇到这个问题，而我需要排除非数据的列，例如标签：

```
> cor(patient.data)
             patient.id         pre     dosage        post
patient.id   1.00000000  0.02286906  0.3643084 -0.13798149
pre          0.02286906  1.00000000  0.2270821 -0.03269263
dosage       0.36430837  0.22708208  1.0000000 -0.42006280
post        -0.13798149 -0.03269263 -0.4200628  1.00000000
```

恼人的是，这个相关矩阵包含无意义的病人身份证号和其他变量的关联。我们可以通过排除病人身份证号列来剔除这些无意义的输出：

```
> cor(subset(patient.data, select = -patient.id))
               pre     dosage        post
pre     1.00000000  0.2270821 -0.03269264
dosage  0.22708207  1.0000000 -0.42006280
post   -0.03269264 -0.4200628  1.00000000
```

我们也可以通过赋予参数select一个否定名称向量来排除多个列：

```
> cor(subset(patient.data, select = c(-patient.id,-dosage)))
             pre        post
pre   1.00000000 -0.03269264
post -0.03269264  1.00000000
```

另请参阅

参见方法5.24更多有关subset函数的内容。

5.29 合并两个数据框

问题

你需要将两个数据框的内容合并成一个数据框。

解决方案

调用cbind函数并排合并两个数据框的列：

```
> all.cols <- cbind(dfrm1,dfrm2)
```

调用rbind函数“堆叠”两个数据框的行：

```
> all.rows <- rbind(dfrm1,dfrm2)
```

讨论

你可以用以下两种方法之一合并数据框：通过将列并排放置来创建一个更宽的数据框；或者通过“堆叠”行来创建一个更长的数据框。函数cbind会并排合并数据框，如下所示合并stooges和birth：

```
> stooges
   name n.marry n.child
1   Moe       1       2
2 Larry       1       2
3 Curly       4       2
> birth
  birth.year  birth.place
1       1887  Bensonhurst
2       1902 Philadelphia
3       1903     Brooklyn
> cbind(stooges,birth)
     name n.marry n.child birth.year  birth.place
1     Moe       1       2       1887  Bensonhurst
2   Larry       1       2       1902 Philadelphia
3   Curly       4       2       1903     Brooklyn
```

你通常合并有着相同高度（行数）的列。然而，理论上讲，函数cbind不需要匹配高度。如果一个数据框很短，它会调用循环规则来尽可能地延伸这个短列（方法5.3），结果不一定如你所愿。

函数rbind会“堆叠”两个数据框的行，如下所示合并stooges和guys：

```
> stooges
   name n.marry n.child
1   Moe       1       2
2 Larry       1       2
3 Curly       4       2
> guys
   name n.marry n.child
1   Tom       4       2
2  Dick       1       4
3 Harry       1       1
> rbind(stooges,guys)
   name n.marry n.child
1   Moe       1       2
2 Larry       1       2
3 Curly       4       2
4   Tom       4       2
5  Dick       1       4
6 Harry       1       1
```

函数rbind需要数据框有相同的宽度：相同的列数和列名。然而，列不需要有相同的顺序（order），rbind会把它们排列出来。

最后，这个方法比标题意指的更宽泛一些。首先，由于rbind和cbind接受多个参数，所以你可以合并多于两个数据框。其次，由于rbind和cbind也适用于向量、列表和矩阵，所以你可以将这个方法应用到其他数据类型。

另请参阅

函数merge可以合并因为包含有缺失或者有差异的列而不兼容的数据框。在CRAN中可用的软件包reshape2和plyr包含一些功能强大的函数用于切片、切块和重组数据框。

5.30 根据共有列合并数据框

问题

你有两个数据框，它们有相同的列。你需要通过匹配这个共有列来合并它们的行，使它们成为一个数据框。

解决方案

调用函数merge，基于共有列把这些数据框连接成一个新数据框：

```
> m <- merge(df1, df2, by="name")
```

这里，*name*是数据框df_1和df_2共有列的列名。

讨论

假设有born和died两个数据框，它们都包含一个称为name的列：

```
> born
   name year.born   place.born
1   Moe      1887 Bensonhurst
2 Larry      1902 Philadelphia
3 Curly      1903     Brooklyn
4 Harry      1964       Moscow
> died
   name year.died
1 Curly      1952
2   Moe      1975
3 Larry      1975
```

我们可以通过参数name来合并匹配的行，把它们合并成一个数据框：

```
> merge(born, died, by="name")
   name year.born   place.born year.died
1 Curly      1903     Brooklyn      1952
2 Larry      1902 Philadelphia      1975
3   Moe      1887  Bensonhurst      1975
```

注意，函数merge不需要行已排序或者以相同顺序出现。它为Curly找到匹配的行，即使它们在不同的位置出现。它同样会摒除只出现在一个或另一个数据框的行。

在结构化查询语言（SQL）的术语中，函数merge实质上对两个数据框执行连接操作。它有许多用于控制连接操作的选项，这些都在函数merge的帮助说明页中有所描述。

另请参阅

参见方法5.29，了解其他合并数据框的方法。

5.31 更便捷地访问数据框内容

问题

你的数据存储在一个数据框中。你厌倦了反复地输入数据框名称，你想要更便捷地访问列。

解决方案

对于快速地、一次性应用数据框内的表达式，可以调用with函数来访问列名：

```
> with(dataframe, expr)
```

在*expr*中，你可以通过名称来访问数据框*dataframe*的列——就像它们是简单变量一样。

如果需要重复访问，你可以调用attach函数在你的搜索列表（search list）中插入数据框。然后你便能够不涉及数据框，仅根据列名称来访问数据框的列：

```
> attach(dataframe)
```

调用detach函数从你的搜索列表中移除数据框。

讨论

数据框是一个存储数据的好方法，但用它来访问单列会变得冗长。对于包含一个列名为pop的名为suburbs的数据框，这里有个原始的方法来计算pop变量的z分数：

```
> z <- (suburbs$pop - mean(suburbs$pop)) / sd(suburbs$pop)
```

然而，这里所有的输入变得冗长乏味。函数with允许你就像访问不同的变量一样来访问数据框的列。它需要两个参数，一个是数据框和一个是要计算的表达式。在表达式中，你可以根据列名称来访问数据框的列：

```
> z <- with(suburbs, (pop - mean(pop)) / sd(pop))
```

如果仅仅应用上述代码一次，这很方便。如果你需要在数据框中反复处理列，则可以将数据框插入到搜索列表中，列名便可以像变量一样来直接访问：

```
> attach(suburbs)
```

在attach函数后，搜索列表中的第二项就是数据框suburbs：

```
> search()
 [1] ".GlobalEnv"        "suburbs"           "package:stats"
 [4] "package:graphics"  "package:grDevices" "package:utils"
 [7] "package:datasets"  "package:methods"   "Autoloads"
[10] "package:base"
```

现在，我们可以把数据框中的列当成变量来访问：

```
> z <- (pop - mean(pop)) / sd(pop)
```

当你结束这段程序时，调用detach函数（无参数）来移除搜索列表中的第二个存储单元（location）：

```
> detach()
> search()
[1] ".GlobalEnv"         "package:stats"   "package:graphics"
[4] "package:grDevices" "package:utils"   "package:datasets"
[7] "package:methods"   "Autoloads"       "package:base"
```

观察到suburbs已经不在搜索列表中了。

插入一个数据框到搜索列表有很大的玄机：R会插入一个数据框的临时副本，这意味着对原始数据框的改变被隐藏了。在下面的代码片段中，注意改变数据框并不改变插入的（attached）数据的视图：

```
> attach(suburbs)
> pop
 [1] 2853114  90352  171782   94487  102746  106221  147779   76031   70834
[10]   72616  74239   83048   67232   75386   63348   91452
> suburbs$pop <- 0           # Overwrite data frame contents
> pop                        # Hey! It seems nothing changed
 [1] 2853114  90352  171782   94487  102746  106221  147779   76031   70834
[10]   72616  74239   83048   67232   75386   63348   91452
> suburbs$pop                # Contents of data frame did indeed change
 [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

引起混淆的另一个来源是将值赋给显示出的名称不一定像你希望的那样运行。在下面的代码片段中，你也许觉得我们以1 000倍的比例缩小pop，然而实际上我们创建了一个新的局部变量：

```
> attach(suburbs)
> pop
 [1] 2853114  90352  171782   94487  102746  106221  147779   76031   70834
[10]   72616  74239   83048   67232   75386   63348   91452
> pop <- pop / 1000          # Achtung! This is creating a local variable called "pop"
> ls()                       # We can see the new variable in our workspace
[1] "pop" "suburbs"
> suburbs$pop                # Original data is unchanged
 [1] 2853114  90352  171782   94487  102746  106221  147779   76031   70834
[10]   72616  74239   83048   67232   75386   63348   91452
```

5.32 基本数据类型之间的转换

问题

你有一个原子数据类型（或称基本数据类型）的数据值：字符型（character）、复数型（complex）、双精度型（double）、整型（integer）或者逻辑型（logical）。你需要将它的值转换成其他基本数据类型的一种。

解决方案

对于每个基本数据类型，都有一个函数用于将其他值转换为这种数据类型。基本类型的转换函数包括：

- as.character(x)
- as.complex(x)
- as.numeric(x) or as.double(x)
- as.integer(x)
- as.logical(x)

讨论

转换基本类型通常很容易。如果转换成功，你能得到你所期望的结果；如果转换不成功，你得到NA值：

```
> as.numeric(" 3.14 ")
[1] 3.14
> as.integer(3.14)
[1] 3
> as.numeric("foo")
[1] NA
Warning message:
NAs introduced by coercion
> as.character(101)
[1] "101"
```

如果你有一个基本类型向量，这些函数会将它们应用于向量的每个元素。所以先前转换纯量的例子很容易地推广到转换整个向量：

```
> as.numeric(c("1","2.718","7.389","20.086"))
[1] 1.000 2.718 7.389 20.086
> as.numeric(c("1","2.718","7.389","20.086", "etc."))
[1] 1.000 2.718 7.389 20.086 NA
Warning message:
NAs introduced by coercion
> as.character(101:105)
[1] "101" "102" "103" "104" "105"
```

在将逻辑值转换为数字值时，R转换FALSE为0，TRUE为1：

```
> as.numeric(FALSE)
[1] 0
> as.numeric(TRUE)
[1] 1
```

当你计算在逻辑值向量中TRUE的出现次数时，这个方法很有用。如果logvec是一个逻辑

值向量，那么sum(logvec)将隐式地把逻辑值转换为整数值，并且返回logvec中TRUE值的个数。

5.33 不同结构化数据类型间的转换

问题

你需要将一个变量从一种结构化数据类型转换成另一种结构化数据类型——例如，将向量转换成列表或者将矩阵转换成数据框。

解决方案

下列函数将它们的参数转换成对应的结构化数据类型：

- as.data.frame(x)
- as.list(x)
- as.matrix(x)
- as.vector(x)

然而，有些转换可能会令你惊奇。我建议你复习表5-1的内容。

讨论

结构化数据类型间的转换会很棘手。有些转换会如你预期的那样进行。例如，如果你要将矩阵转换成数据框，矩阵的行和列会轻而易举地变成数据框的行和列。在另一些情况下，结果也许令你吃惊。表 5-1总结了一些值得注意的例子。

表5-1：数据转换

转换	转换方法	备注
向量到列表	as.list(vec)	不能用list(vec)，它将建立含有一个元素的列表，该元素就是向量vec
向量到矩阵	建立1列矩阵：cbind(vec)或as.matrix(vec) 建立1行矩阵:rbind(vec) 建立$n \times m$矩阵：matrix(vec,n,m)	参见方法5.14
向量到数据框	建立1列数据框：as.data.frame(vec) 建立1行数据框：as.data.frame(rbind(vec))	

表5-1：数据转换（续）

转换	转换方法	备注
列表到向量	unlist(lst)	用unlist而不用as.vector，参见方法5.11和注记1
列表到矩阵	建立1列矩阵：as.matrix(lst) 建立1行矩阵：as.matrix(rbind(lst)) 建立$n \times m$矩阵：matrix(lst,n,m)	
列表到数据框	列表元素作为数据框列：as.data.frame(lst); 列表元素作行：参考方法5.19	
矩阵到向量	as.vector(mat)	矩阵所有元素作为向量元素
矩阵到列表	as.list(mat)	所有矩阵元素存于一个列表中
矩阵到数据框	as.data.frame(mat)	
数据框到向量	对于1行的数据框：dfrm[1,] 对于1列的数据框：dfrm[,1]或dfrm[[1]]	参见注记2
数据框到列表	as.list(dfrm)	参见注记3
数据框到矩阵	as.matrix(dfrm)	参见注记4

下列注记被引用在表格中：

注记1. 当把列表转换成向量时，如果列表包含相同模式的单元值，那么转换会清晰地进行。如果遇到下列情况之一，事情将变得很复杂：1）列表中包含混合模式（例如，数值型和字符型），这种情况下任何值都要转换成字符型，2）列表包含其他结构化数据类型，例如子列表或数据框——在这种情况下，奇怪的事情会发生，所以不要这么做。

注记2. 只有当数据框包含一行或者一列时，将数据框转换成向量才有意义。为了抽取它所有的元素使其成为一个长向量，调用函数as.vector(as.matrix(dfrm))。但只有数据框是全数值型或者全字符型时，这才有意义；如果情况并非如此，所有值会先转换成字符串。

注记3. 由于数据框已经是列表（例如，包含列的列表），将数据框转换成列表会看似奇怪。调用函数as.list从本质上移除类（data.frame），因此显示出了隐含的列表。当需要R把数据结构看做列表时（例如为了输出它），这会很有用。

注记4. 小心将数据框转换成矩阵。如果数据框仅包含数值，那么将得到一个数值型矩阵。如果它仅包含字符值，那么你得到一个字符型矩阵。但如果数据框是一个

数字的混合（字符型和因子），那么所有值首先转换成字符型。最终结果将是字符串矩阵。

矩阵的问题

矩阵转换的前提是矩阵是同质的（所有元素类型相同）：所有元素有相同的模式（例如，所有为数值型或者所有为字符型）。当矩阵由列表构成时，它也可以是异质的。如果这样的话，转换将变得凌乱。例如，当你将一个混合模式矩阵转换成一个数据框时，数据框的列实际上是列表（用于容纳混合数据）。

另请参阅

参见方法5.32关于转换基本数据类型的内容；参见本章“简介”对存在问题的转换的评论。

第6章

数据转换

简介

本章主要讲述apply系列函数：apply、lapply、sapply、tapply、mapply，以及它们的姊妹函数，by和split。这些函数允许你大量地处理数据并且一次性完成。传统的编程语言使用循环（loop），而R使用向量化运算和apply系列函数来批量处理数据，极大地促进了计算的流线化。

通过因子定义组

R的一个重要术语是运用因子（factor）来定义组。假设我们有相同长度的一个向量和一个因子，它们可以用如下来创建：

```
> v <- c(40,2,83,28,58)
> f <- factor(c("A","C","A","B","C"))
```

我们可以并排地可视化向量元素和因子水平，如下图所示。

向量	因子
40	A
2	C
83	A
28	B
58	C

因子水平识别每个向量元素的组：40和83在A组；28在B组；2和58在C组。

在本书中，把这些因子称为分组因子（grouping factor）。通过把数据放入组，可以有效地对它们进行切片和切块。它的强大之处在于，要分析的统计数据经常以组为单位出现，例如比较组均值、比较组比例、进行方差分析等。

本章包含使用分组因子将向量元素划分到各自所在组的方法（参见方法6.1），把函数应用于一个向量的各个组（参见方法6.5），以及将函数应用于一个数据框的行组（参见方法6.6）。在其他章中，相同的术语用于检验组均值（参见方法9.19），进行单向方差分析（参见方法11.20），根据组来绘制数据点（参见方法10.4），以及其他用途等。

6.1 向量分组

问题

你有一个向量，其中每个元素属于不同的组，并且组由一个分组因子识别。你需要将这些元素分组。

解决方案

假设向量是x，因子是f。可以调用split函数：

```
> groups <- split(x, f)
```

另外，也可以调用unstack函数：

```
> groups <- unstack(data.frame(x,f))
```

两者都返回一个向量列表，列表中的每个向量包含属于一个组的元素。

函数unstack有更深一层的作用：如果所有向量有相同的长度，它会将列表转换成一个数据框。

讨论

数据集Cars93包含一个称为Origin的因子，它有两个水平，USA和non-USA。它也包含一个叫做MPG.city的列。我们可以根据Origin因子对MPG数据分组，如下所示。

```
> library(MASS)
> split(Cars93$MPG.city, Cars93$Origin)
$USA
 [1] 22 19 16 19 16 16 25 25 19 21 18 15 17 17 20 23 20 29 23 22 17 21 18 29 20
[26] 31 23 22 22 24 15 21 18 17 18 23 19 24 23 18 19 23 31 23 19 19 19 28
```

```
$non-USA
 [1] 25 18 20 19 22 46 30 24 42 24 29 22 26 20 17 18 18 29 28 26 18 17 20 19 29
[26] 18 29 24 17 21 20 33 25 23 39 32 25 22 18 25 17 21 18 21 20
```

分组的作用在于我们能够以小组为单位分析数据。下面的例子计算了每一组MPG的中位数：

```
> g <- split(Cars93$MPG.city, Cars93$Origin)
> median(g[[1]])
[1] 20
> median(g[[2]])
[1] 22
```

另请参阅

参见本章“简介”中关于分组因子的定义。方法6.5展示了另一个应用函数的方法，例如对于每一组，调用median。除上面的功能之外，函数unstack可以执行其他更多的强大转换；参见帮助说明页。

6.2 将函数应用于每个列表元素

问题

你有一个列表，并且需要对这个列表中的每个元素应用一个函数。

解决方案

调用函数lapply或者sapply，这取决于结果所需要的形式。函数lapply通常返回列表形式的结果。然而有可能的话，函数sapply会返回向量形式的结果：

```
> lst <- lapply(lst, fun)
> vec <- sapply(lst, fun)
```

讨论

对于列表中的每个元素，这些函数都会调用一次函数（例如，解决方案中的*fun*函数）。函数期望有一个参数，这个参数是列表中的一个元素。函数lapply和sapply会收集返回值。lapply将它们收集到一个列表中并且返回这个列表。

函数“sapply”中的“s”代表“简化”（simplify）。这个函数试图将结果简化成一个向量或者矩阵。为了让这个结果发生，所有返回值必须有相同的长度。如果长度为1，那么会得到一个向量；否则，会得到一个矩阵。如果长度不一样，便不可能发生简化，就会得到一个列表。

比如，我教一门介绍性的统计学课程四次，并且每次都对期末考试结果进行整理。下面是这四个学期的考试成绩：

```
> scores
$S1
[1] 89 85 85 86 88 89 86 82 96 85 93 91 98 87 94 77 87 98 85 89
[21] 95 85 93 93 97 71 97 93 75 68 98 95 79 94 98 95

$S2
[1] 60 98 94 95 99 97 100 73 93 91 98 86 66 83 77
[16] 97 91 93 71 91 95 100 72 96 91 76 100 97 99 95
[31] 97 77 94 99 88 100 94 93 86

$S3
[1] 95 86 90 90 75 83 96 85 83 84 81 98 77 94 84 89 93 99 91 77
[21] 95 90 91 87 85 76 99 99 97 97 97 77 93 96 90 87 97 88

$S4
[1] 67 93 63 83 87 97 96 92 93 96 87 90 94 90 82 91 85 93 83 90
[21] 87 99 94 88 90 72 81 93 93 94 97 89 96 95 82 97
```

每个学期开始时都有40个学生，然而，并非每个人都能坚持学到学期结束，因此每个学期都有不同数量的学生成绩。我们可以调用length函数计算成绩的数量：lapply会返回一个长度列表，sapply会返回一个长度向量，例如：

```
> lapply(scores, length)
$S1
[1] 36

$S2
[1] 39

$S3
[1] 38

$S4
[1] 36

> sapply(scores, length)
S1 S2 S3 S4
36 39 38 36
```

我们可以很方便地看到这些成绩的均值和标准差：

```
> sapply(scores, mean)
      S1       S2       S3       S4
88.77778 89.79487 89.23684 88.86111
> sapply(scores, sd)
       S1        S2       S3       S4
 7.720515 10.543592 7.178926 8.208542
```

如果调用的函数返回的是一个向量，sapply会将结果转换为一个矩阵。例如，函数range返回一个二元素的向量：

```
> sapply(scores, range)
     S1  S2 S3 S4
[1,] 68  60 75 63
[2,] 98 100 99 99
```

如果调用的函数返回一个结构化对象，例如一个列表，那么你需要使用lapply而不是sapply。结构化对象不能放入向量中。假设我们需要对每个学期执行t检验。函数t.test返回一个列表，所以我们必须调用lapply：

```
> tests <- lapply(scores, t.test)
```

现在得到的结果tests是一个列表：它是一个t.test结果的列表。我们可以调用sapply从t.test的结果中提取相关的元素，例如t检验置信区间的边界值：

```
> sapply(tests, function(t) t$conf.int)
           S1       S2       S3       S4
[1,] 86.16553 86.37703 86.87719 86.08374
[2,] 91.39002 93.21271 91.59650 91.63848
```

另请参阅

参见方法2.12。

6.3 将函数应用于每行

问题

你有一个矩阵。你需要将函数应用于每一行，对每一行计算函数的结果。

解决方案

调用函数apply。设置第二个参数为1，它表明逐行来应用一个函数：

```
> results <- apply(mat, 1, fun)     # mat is a matrix, fun is a function
```

函数apply会对mat的每一行调用一次fun，将返回值存储到一个向量中，然后返回这个向量。

讨论

假设矩阵long是纵向数据。每一行包含一个观测对象数据，并且它的列包含随时间重复的观测值：

```
> long
```

```
        trial1    trial2     trial3     trial4     trial5
Moe -1.8501520 -1.406571 -1.0104817 -3.7170704 -0.2804896
Larry  0.9496313  1.346517 -0.1580926  1.6272786  2.4483321
Curly -0.5407272 -1.708678 -0.3480616 -0.2757667 -1.2177024
```

你可以通过将mean函数应用于每行来计算每个观测对象的平均观测值，最终结果是一个向量：

```
> apply(long, 1, mean)
       Moe      Larry      Curly
-1.6529530 1.2427334 -0.8181872
```

注意，函数apply使用矩阵数据中的rownames来标识结果向量的元素，这很容易得到。

这个被调用的函数（正如先前描述的函数fun）会期望一个参数，即一个向量值，它就是矩阵中的一行。函数会返回一个纯量或者一个向量。在返回向量的情况下，函数apply将结果存储到一个矩阵中。函数range返回一个二元素的向量，它们是最小值和最大值，所以将它应用于long会产生一个矩阵：

```
> apply(long, 1, range)
            Moe       Larry      Curly
[1,] -3.7170704  -0.1580926  -1.7086779
[2,] -0.2804896   2.4483321  -0.2757667
```

也可以将本节的方法应用于数据框。如果数据框是同质的——所有元素为数字或者为字符串，它就会奏效。当数据框有不同类型的列时，由于向量必须是同质的，所以从每一行提取向量是没有任何意义的。

6.4 将函数应用于每列

问题

你有一个数据框或者矩阵，需要对每一列应用一个函数。

解决方案

对于矩阵，使用apply函数。设置第二个参数为2，它表明逐列应用该函数：

```
> results <- apply(mat, 2, fun)
```

对于数据框，使用lapply或者sapply函数。它们都会把函数应用于数据框的连续列。它们的区别是，lapply把返回值存储到一个列表，然而sapply把它们存储到一个向量：

```
> lst <- lapply(dfrm, fun)
> vec <- sapply(dfrm, fun)
```

你也可以对数据框使用apply，但只有当数据框是同质时（即，元素全为数值或者字符串）才可以。

讨论

函数apply是用来处理矩阵的。在方法6.3中，我们调用apply来处理一个矩阵的行。这里的情况是一样的，只是现在我们处理列。函数apply的第二个参数决定了方向：

- 参数为1意味着逐行处理。
- 参数为2意味着逐列处理。

这比它看起来更有助于记忆。我们以"行和列"论及矩阵，所以行是第一而列为第二；因此，表示它们的参数分别为1和2。

数据框是比矩阵更加复杂的数据结构，所以它有更多的选项。你可以简单地使用apply，在这种情况下，R会把数据框转换为一个矩阵然后应用函数。如果数据框仅包含数字数据或者仅包含字符数据，这便会成功运行。如果有混合的数据类型，它可能就不起作用。在那种情况下，R会强行把所有列转换成相同类型，结果是很可能会执行一个不必要的转换。

幸运的是，这里有其他方案选择。前面讲过数据框是列表的一种：它是数据框的列的一个列表。可以使用lapply和sapply来处理列，如方法6.2所描述：

```
> lst <- lapply(dfrm, fun)          # Returns a list
> vec <- sapply(dfrm, fun)          # Returns a vector
```

函数fun应该预期一个参数：数据框中的一列。

我通常使用这个方法来检查数据框中列的类型。以下数据框的batch列看似包含数字：

```
> head(batches)
  batch clinic dosage  shrinkage
1     1     IL      3 -0.11810714
2     3     IL      4 -0.29932107
3     2     IL      4 -0.27651716
4     1     IL      5 -0.18925825
5     2     IL      2 -0.06804804
6     3     NJ      5 -0.38279193
```

但是，列类型的输出结果却表明它是一个因子：

```
> sapply(batches, class)
    batch    clinic    dosage shrinkage
 "factor"  "factor" "integer" "numeric"
```

应用本节方法的一个很好的例子是从一组预测变量中剔除低相关的预测变量。假设resp是响应变量（即因变量），而pred是一个包含预测变量的数据框。进一步假设我们有过多的预测变量，因此需要选定前10个与响应变量相关性最大的预测变量。

第一步是计算每个变量与resp之间的相关系数。在R中，就一行简单代码：

```
> cors <- sapply(pred, cor, y=resp)
```

函数sapply会调用cor函数处理在pred中的每一列。注意，对于sapply，给出了第三个参数，y=resp。任何第二个参数之后的都传给函数cor。每次sapply调用cor时，第一个参数是一列而第二个参数是y=resp。有了这些参数，函数调用会是cor(*column*,y=*resp*)，它计算给定列与resp的相关系数。

函数sapply返回的结果是一个相关系数向量，每个相关系数对应一列。调用rank函数找到有最大量级的相关系数的位置：

```
> mask <- (rank(-abs(cors)) <= 10)
```

这个表达式是一个比较关系（<=），所以它返回一个逻辑值向量。它构造清晰，所以前10个相关系数有相应的TRUE值，并且其他相关系数都为FALSE值。使用这个逻辑值函数，可以从这个数据框中仅仅选定这些有TRUE值的列：

```
> best.pred <- pred[,mask]
```

此时，我们已经选择了与resp相关性最高的预测变量，可以用best.pred对resp进行回归：

```
> lm(resp ~ best.pred)
```

上面四行代码就完成了变量选择，十分方便。

另请参阅

参见方法5.22、方法6.2和方法6.3。

6.5 将函数应用于组数据

问题

数据元素以小组为单位呈现。你需要按分组处理数据——例如，按组求和或者按组求平均值。

解决方案

创建一个分组因子（与向量有相同的长度）识别每个对应数据的组。然后调用tapply函数，它会将一个函数应用于每组数据：

```
> tapply(x, f, fun)
```

这里，*x*是一个向量，*f*是一个分组因子，*fun*是一个函数。函数应该预计一个参数，它是一个根据分组取自*x*的向量元素。

讨论

假设有一个向量，它包含芝加哥大都会区最大的16个城市的人口数据，取自名为suburbs的数据框：

```
> attach(suburbs)
> pop
 [1] 2853114 90352 171782 94487 102746 106221 147779 76031 70834
[10]   72616 74239  83048 67232  75386  63348  91452
```

可以容易地对所有这些城市计算出总数和平均值：

```
> sum(pop)
[1] 4240667
> mean(pop)
[1] 265041.7
```

如果我们需要每个郡的人口总数和平均值，该如何处理呢？我们需要一个因子，即郡名（county），与人口（pop）有相同的长度，然后把郡名因子的每个水平赋予相应的城市（其中有两个湖郡：一个位于伊利诺斯州（IL），另一个位于印第安纳州（IN））：

```
> county
 [1] Cook     Kenosha  Kane     Kane     Lake(IN) Kendall  DuPage   Cook
 [9] Will     Cook     Cook     Lake(IN) Cook     Cook     Cook     Lake(IL)
Levels: Cook DuPage Kane Kendall Kenosha Lake(IL) Lake(IN) Will
```

现在可以使用因子county和函数tapply以组为单位来处理。函数tapply有三个主要参数：数据向量、定义组的因子和一个函数。它会提取每组，将函数应用于每一组，并且返回一个包含收集结果的向量。下面的例子显示根据郡的人口求和：

```
> tapply(pop,county,sum)
   Cook  DuPage    Kane Kendall Kenosha Lake(IL) Lake(IN)    Will
3281966  147779  266269  106221   90352    91452   185794   70834
```

下一个例子根据郡计算平均人口：

```
> tapply(pop,county,mean)
```

```
    Cook   DuPage     Kane  Kendall  Kenosha Lake(IL) Lake(IN)     Will
468852.3 147779.0 133134.5 106221.0  90352.0  91452.0  92897.0  70834.0
```

tapply预期一个参数：一个包含一组中所有元素的向量。一个很好的例子是length函数，它接受一个向量参数并返回该向量的长度。我们可以使用它计算每一组中的数据的数目，因此，每个郡的城市数为：

```
> tapply(pop,county,length)
    Cook   DuPage     Kane  Kendall  Kenosha Lake(IL) Lake(IN)     Will
       7        1        2        1        1        1        2        1
```

在多数情况下，你会调用返回一个纯量的函数，并且tapply会收集返回的纯量并放入一个向量中。函数也可以返回复杂的对象，在这种情况下，tapply会以列表形式返回它们。有关详细信息参见tapply的帮助说明页。

另请参阅

参见本章“简介”中更多关于分组因子的内容。

6.6 将函数应用于行组

问题

需要将函数应用于数据框中的行组。

解决方案

定义一个分组因子（数据框中的每一行有一个相应的因子水平），识别该行所属的组别。

对于每一个这样的行组，函数by把行放入一个临时的数据框，并且用它为参数调用函数。函数by把返回值存储在列表中，并返回这个列表：

```
> by(dfrm, fact, fun)
```

这里，*dfrm*是数据框，*fact*是分组因子，*fun*是一个函数。函数*fun*应该预期一个数据框参数。

讨论

函数by的优势在于它用一个数据框调用函数，如果函数以一种特殊的方式处理数据框，这就会很有用。例如，函数print、summary以及mean对数据框执行特殊处理。

假设你从临床实验中得到一个数据框，称为trials，它的剂量被随机化以研究它的效果：

```
> trials
  sex      pre dose1 dose2     post
1   F 5.931640     2     1 3.162600
2   F 4.496187     1     2 3.293989
3   M 6.161944     1     1 4.446643
4   F 4.322465     2     1 3.334748
5   M 4.153510     1     1 4.429382
.
. (etc.)
.
```

该数据包含一个因子来识别每个记录的性别，所以by可以根据性别将数据分为2组，并且对这两组分别调用summary函数。结果为两个汇总，一个是关于男性的，另一个是关于女性的：

```
> by(trials, trials$sex, summary)
trials$sex: F
sex      pre            dose1           dose2            post
F:7  Min.   :4.156  Min.   :1.000  Min.   :1.000  Min.   :2.886
M:0  1st Qu.:4.409  1st Qu.:1.000  1st Qu.  :1.000  1st Qu  :3.075
     Median :4.895  Median :1.000  Median   :2.000  Median  :3.163
     Mean   :5.020  Mean   :1.429  Mean     :1.571  Mean    :3.174
     3rd Qu.:5.668  3rd Qu.:2.000  3rd Qu.  :2.000  3rd Qu. :3.314
     Max.   :5.932  Max.   :2.000  Max.     :2.000  Max.    :3.389
---------------------------------------------------------------------
trials$sex: M
sex      pre            dose1           dose2            post
F:0  Min.   :3.998  Min.   :1.000  Min.     :1.000  Min.    :3.738
M:9  1st Qu.:4.773  1st Qu.:1.000  1st Qu.  :1.000  1st Qu. :3.800
     Median :5.110  Median :2.000  Median   :1.000  Median  :4.194
     Mean   :5.189  Mean   :1.556  Mean     :1.444  Mean    :4.148
     3rd Qu.:5.828  3rd Qu.:2.000  3rd Qu.  :2.000  3rd Qu. :4.429
     Max.   :6.658  Max.   :2.000  Max.     :2.000  Max.    :4.517
```

我们也可以把post作为剂量的函数，据此构建两个线性模型，一个模型针对男性，另一个描述女性：

```
> models <- by(trials, trials$sex, function(df) lm(post~pre+dose1+dose2, data=df))
```

注意，函数的参数是一个数据框，所以我们可以用它作为函数lm的数据参数。结果为一个2元素的线性模型列表。当输出这个列表时，我们得到每个性别的模型：

```
> print(models)
trials$sex: F

Call:
lm(formula = post ~ pre + dose1 + dose2, data = df)
```

```
Coefficients:
(Intercept)          pre        dose1        dose2
    4.30804     -0.08161     -0.16225     -0.31354

--------------------------------------------------------
trials$sex: M

Call:
lm(formula = post ~ pre + dose1 + dose2, data = df)

Coefficients:
(Intercept)          pre        dose1        dose2
    5.29981     -0.02713     -0.36851     -0.30323
```

我们有一个模型列表，所以可以对每个列表元素调用confint函数，得到每个模型系数的置信区间：

```
> lapply(models, confint)
$F
                 2.5 %      97.5 %
(Intercept)  3.0841733  5.53191431
pre         -0.2950747  0.13184560
dose1       -0.4711773  0.14667409
dose2       -0.6044273 -0.02264593
$M
                 2.5 %      97.5 %
(Intercept)  4.8898433  5.70978218
pre         -0.1070276  0.05277108
dose1       -0.4905828 -0.24644057
dose2       -0.4460200 -0.16043211
```

在这种情况下，我们发现两个模型中pre的置信区间包含零，所以它对任何一个模型都没有意义；与此相反，dose1和dose2对两个模型都有意义。然而，关键的事实是两个模型的截距有显著差异，这提醒我们对男性和女性的响应可能不同。

另请参阅

参见本章“简介”中更多关于分组因子的内容。参见方法6.2更多关于lapply函数的内容。

6.7 将函数应用于平行向量或列表

问题

你有一个函数f，它有多个参数。你需要将这个函数应用于向量的每个元素，并且得到一个向量结果。不幸的是，函数不是向量化的，即它以纯量而非向量的形式运行。

解决方案

调用mapply函数，它将函数f应用于参数的每个元素：

```
> mapply(f, vec₁, vec₂, ..., vecₙ)
```

这里，函数f预期的每个参数必须为一个向量。如果向量参数有不等的长度，则调用循环规则。

函数mapply也处理列表参数：

```
> mapply(f, list₁, list₂, ..., listₙ)
```

讨论

R的基本运算符号，例如$x + y$，是向量化的，这意味着它们按逐个元素计算结果并且返回一个向量结果。同样，许多R函数是向量化的。

然而，不是所有函数都是向量化的，而且它们也不仅仅作用于纯量。应用向量参数可能会产生错误，或者产生没有意义的结果。在这种情况下，函数mapply可以有效地向量化函数。

考虑方法2.12的函数gcd，它有两个参数：

```
> gcd <- function(a,b) {
+       if (b == 0) return(a)
+       else return(gcd(b, a %% b))
+ }
```

如果我们用两个向量来调用函数gcd，结果是错误的并且会显示很多错误信息：

```
> gcd(c(1,2,3), c(9,6,3))
[1] 1 2 0
Warning messages:
1: In if (b == 0) return(a) else return(gcd(b, a%%b)) :
  the condition has length > 1 and only the first element will be used
2: In if (b == 0) return(a) else return(gcd(b, a%%b)) :
  the condition has length > 1 and only the first element will be used
3: In if (b == 0) return(a) else return(gcd(b, a%%b)) :
  the condition has length > 1 and only the first element will be used
```

这个函数不是向量化的，但我们可以调用mapply将它向量化，从而可以求两个向量相应元素对的最大公约数（GCD）：

```
> mapply(gcd, c(1,2,3), c(9,6,3))
[1] 1 2 3
```

第7章

字符串和日期

简介

字符串？日期？在统计软件包中？

只要读取文件或打印报告，你就需要字符串。当你处理现实世界的问题时，你就需要日期。

R具备处理字符串和日期的功能。尽管R的字符串功能与面向字符串的语言（如Perl语言）相比显得笨拙，但是这是一个不同工作选择不同的适合工具的问题。我不会在Perl中执行逻辑回归。

日期和时间类

R有各种处理日期和时间的类。如果你喜欢有多种选择，这是很好的；但如果你喜欢生活简单，这是恼人的。类之间有一个关键的区别：有些类是只针对日期的类，有些类是日期时间类。所有类都可以处理日历日期（例如，2010年3月15日），但不是所有类都可以表示一个日期时间（2010年3月1日上午11:45）。

R的基本发行版包括以下类：

`Date`类

Date类可以代表一个日历日期，但不是一个时钟时间。对于处理日期，它是一个坚实的、通用的类，它包括转换、格式化、基本日期算法、时区处理。本书中与大多数日期相关的方法都建立在Date类上。

POSIXct类

这是一个日期时间类，它可以表示时间上的某个时刻，能精确到秒。在R内部，日期时间是存储从1970年1月1日起到该时间的以秒计的一个数值，它是一个非常紧凑的表示。建议用这个类存储日期时间信息（例如，在数据框中）。

POSIXlt类

这也是一个日期时间类，但它的表示存储在一个9元素列表中，其中包括年、月、日、小时、分钟和秒。这个表示可以很容易地提取部分日期元素，如月或小时。显然，这个表示没有POSIXct类紧凑，因此它通常用于中间处理而非存储数据。

基础发布版还提供了在不同日期表示之间轻松转换的函数，例如as.Date、as.POSIXct、as.POSIXlt。

以下R软件包可以从CRAN上下载：

chron

chron可以表示日期和时间，但没有增加足以处理时区和夏令时的复杂性。因此它比Date类更容易使用，但没有POSIXct类和POSIXlt类强大。它适合用于计量经济学和时间序列分析中。

lubridate

这是一个相对较新的、通用的R包。它使处理日期和时间工作更简单，同时保持了重要的功能，如处理时区。它特别适用于处理日期时间算法。

mondate

除了可以处理日和年以外，这个包特别适合于处理以月为单位的日期。在会计和精算工作中会出现这样的需求，例如需要按月进行计算。

timeDate

这是一个深思熟虑的用于处理日期和时间的功能强大的软件包，它包括日期算法、工作日、节假日、转换和广义的时区处理。它原来是Rmetrics软件的一部分，用于金融建模，金融模型中对日期和时间的精度要求是至关重要的。如果你对日期功能要求苛刻，可以考虑使用这个软件包。

应该选择哪一个日期和时间类？Grothendieck和Petzoldt的文章“Date and Time Classes in R”提供以下指导意见：

> 考虑使用哪个类时，总是选择支持你的应用的最不复杂的类。也就是说，尽可能地使用Date，否则使用chron类，然后考虑POSIX类。这种策略将大大降低潜在犯错的可能，并增加应用程序的可靠性。

另请参阅

参见help(DateTimeClasses)给出的文档，来了解更多关于内置功能的详细信息。参见2004年6月Grothendieck和Petzoldt的文章“Date and Time Classes in R”（*http://cran.r-project.org/doc/Rnews/Rnews_2004-1.pdf*），它是关于日期和时间功能的很好的介绍。2001年6月，Brian Ripley和Kurt Hornik的文章“Date-Time Classes”（*http://cran.r-projext.org/doc/Rnews/Rnews_2012-2.pdf*）着重讨论了两个POSIX类。

7.1 获取字符串长度

问题

你想知道一个字符串的长度。

解决方案

调用函数nchar，而非函数length。

讨论

函数nchar以一个字符串为参数，返回这个字符串中的字符数：

```
> nchar("Moe")
[1] 3
> nchar("Curly")
[1] 5
```

如果对一个字符串向量运用nchar，它返回每个字符串的长度：

```
> s <- c("Moe", "Larry", "Curly")
> nchar(s)
[1] 3 5 5
```

你可能认为函数length返回一个字符串的长度。其实不然——它返回一个向量的长度。当对单个字符串运用函数length时，R返回值为1，因为它把这个字符串作为一个向量看待——一个只有一个元素的向量：

```
> length("Moe")
[1] 1
> length(c("Moe","Larry","Curly"))
[1] 3
```

7.2 连接字符串

问题

你想要把两个或者多个字符串连接成一个字符串。

解决方案

调用函数paste。

讨论

函数paste连接多个字符串。换句话说，它通过首尾连接给定的字符串，从而创建一个新的字符串：

```
> paste("Everybody", "loves", "stats.")
[1] "Everybody loves stats."
```

默认情况下，函数paste在一对字符串中间插入一个空格，如果这是你想要的结果，那就很便捷，否则很恼人。参数sep允许你指定一个不同的分隔符。例如用一个空字符串（""）作为分隔符，它使字符串没有间隔地连接在一起：

```
> paste("Everybody", "loves", "stats.", sep="-")
[1] "Everybody-loves-stats."
> paste("Everybody", "loves", "stats.", sep="")
[1] "Everybodylovesstats."
```

该函数对于非字符串参数兼容。它尝试通过调用函数as.character把它们转换成字符串：

```
> paste("The square root of twice pi is approximately", sqrt(2*pi))
[1] "The square root of twice pi is approximately 2.506628274631"
```

如果一个或者更多个参数是字符串向量，则paste会生成所有参数的组合：

```
> stooges <- c("Moe", "Larry", "Curly")
> paste(stooges, "loves", "stats.")
[1] "Moe loves stats." "Larry loves stats." "Curly loves stats."
```

有时你想要把这些组合连接成一个大的字符串。参数collapse让你定义一个顶级分隔符并且指示paste使用该分隔符连接新生成的字符串：

```
> paste(stooges, "loves", "stats", collapse=", and ")
[1] "Moe loves stats, and Larry loves stats, and Curly loves stats"
```

7.3 提取子串

问题

你想要根据位置提取一个字符串的一部分。

解决方案

调用函数substr(*string*,*start*,*end*)提取开始于*start*，结束于*end*的子串。

讨论

函数substr的参数分别是一个原始的字符串、一个起始点和一个结束点。它返回起始点和结束点之间的子串：

```
> substr("Statistics", 1, 4)     # Extract first 4 characters
[1] "Stat"
> substr("Statistics", 7, 10)    # Extract last 4 characters
[1] "tics"
```

像许多R函数一样，substr允许第一个参数为字符串向量。在这种情况下，它会应用向量中的每个字符串元素并且返回一个子串向量：

```
> ss <- c("Moe", "Larry", "Curly")
> substr(ss, 1, 3)                  # Extract first 3 characters of each string
[1] "Moe" "Lar" "Cur"
```

事实上，所有的参数都可以是向量，在这种情况下，substr把它们视为平行向量。从每个字符串中，它根据相应的起始点和结束点来提取子串。这里有一些有用的技巧。例如，下面的代码片段从每个字符串中提取最后两个字符，每个子串起始于原始字符串的倒数第二个字符，结束于最后一个字符：

```
> cities <- c("New York, NY", "Los Angeles, CA", "Peoria, IL")
> substr(cities, nchar(cities)-1, nchar(cities))
[1] "NY" "CA" "IL"
```

你可以利用循环规则拓展这个方法，但我建议你避免这么做。

7.4 根据分隔符分割字符串

问题

你要将一个字符串分割为多个子串，子串之间用分隔符分开。

解决方案

调用strsplit，它需要两个参数——字符串和子串的分隔符：

```
> strsplit(string, delimiter)
```

*delimiter*可以是一个简单的字符串或者一个正则表达式。

讨论

字符串包含由相同的分隔符分割的多个子串是很普遍的。一个例子是一个文件路径，它的各个组成部分由斜杠（“/”）分隔：

```
> path <- "/home/mike/data/trials.csv"
```

通过调用strsplit和分隔符“/”，我们可以把这个路径分割成多个组成部分：

```
> strsplit(path, "/")
[[1]]
[1] ""  "home"  "mike"  "data"  "trials.csv"
```

注意，第一个“组件”实际上是一个空字符串，因为第一个斜杠前面没有内容。

还注意到，strsplit返回一个列表，列表中的每个元素是一个子串向量。这种两级结构是必要的，因为第一个参数可以是一个字符串向量。每个字符串分为多个子串（向量），然后这些向量以一个列表的方式返回。下面的例子分割三个文件路径，然后返回一个3元素的列表：

```
> paths <- c("/home/mike/data/trials.csv",
+            "/home/mike/data/errors.csv",
+            "/home/mike/corr/reject.doc")
> strsplit(paths, "/")
[[1]]
[1] ""  "home"  "mike"  "data"  "trials.csv"
[[2]]
[1] ""  "home"  "mike"  "data"  "errors.csv"
[[3]]
[1] ""  "home"  "mike"  "corr"  "reject.doc"
```

函数strsplit的第二个参数（即分隔符参数）实际上比这些例子中演示的功能更加强大。它可以是一个正则表达式，匹配比简单字符串更加复杂的模式。事实上，要消除正则表达式功能（和它对特殊字符的解释），必须包括参数fixed=TRUE。

另请参阅

要了解更多有关R中的正则表达式，参见函数regexp的帮助说明页。参见由Jeffrey

E.F.撰写的《Mastering Regular Expressions》（O'Reilly）（*http://oreilly.com/catalog/9780596528126/*），以了解更多有关正则表达式的内容。

7.5 替代子串

问题

在一个字符串中，你想要用一个子串替代另一个子串。

解决方案

调用sub来替代字符串的第一个子串实例：

```
> sub(old, new, string)
```

调用gsub来替代字符串的所有子串实例：

```
> gsub(old, new, string)
```

讨论

函数sub找到*string*中子串*old*的第一个实例，然后把它替代为子串*new*：

```
> s <- "Curly is the smart one. Curly is funny, too."
> sub("Curly", "Moe", s)
[1] "Moe is the smart one. Curly is funny, too."
```

gsub同样如此，但它替代字符串中的所有子串*old*的实例（全局替代），不只是第一个：

```
> gsub("Curly", "Moe", s)
[1] "Moe is the smart one. Moe is funny, too."
```

为了从字符串中移除一个子串，可以简单地设定子串*new*为空：

```
> sub(" and SAS", "", "For really tough problems, you need R and SAS.")
[1] "For really tough problems, you need R."
```

参数*old*可以是正则表达式，它允许匹配比简单字符串更复杂的模式。它是默认的设置，所以如果不希望sub和gsub把*old*解释为正则表达式，必须设定参数fixed=TRUE。

另请参阅

要学习更多关于R的正则表达式的内容，可参见regexp的帮助说明页。参见《Mastering Regular Expressions》一书可得到更多关于正则表达式的内容。

7.6 查看字符串中的特殊字符

问题

你的字符串包含不能输出的特殊字符。你想要知道这些特殊字符是什么。

解决方案

调用print查看一个字符串中的特殊字符。函数cat不会显示它们。

讨论

我曾经遇到一个奇怪的情况，在一个字符串中包含我不认识也不可输出的字符，我的R脚本使用cat显示字符串，直到我意识到有不可输出字符在字符串中之前，脚本的输出是乱码。

在这个例子中，字符串似乎包含13个字符，但cat只显示6个字符的输出：

```
> nchar(s)
[1] 13
> cat(s)
second
```

原因是该字符串包含特殊字符，它在输出时不显示。当我们使用print时，它使用转义符（反斜杠）显示特殊字符：

```
> print(s)
[1] "first\rsecond\n"
```

注意，该字符串在中间包含一个回车符（\r），所以当cat显示字符串时，“first”被“second”覆盖了。

7.7 生成字符串的所有成对组合

问题

你有两组字符串，而你需要生成这两组字符串的所有组合（即它们的笛卡儿积）。

解决方案

调用函数outer和paste生成所有可能组合的矩阵：

```
> m <- outer(strings1, strings2, paste, sep="")
```

讨论

函数outer的目的是形成外积。然而，它允许第三个参数用任何函数来代替简单的乘法。在这个方法中，我们用字符串连接（paste）取代乘法，结果是字符串的所有组合。

假设你有4个试验场地和3种处理：

```
> locations <- c("NY", "LA", "CHI", "HOU")
> treatments <- c("T1", "T2", "T3")
```

我们可以运用outer和paste生成试验场地和处理的所有组合：

```
> outer(locations, treatments, paste, sep="-")
     [,1]     [,2]     [,3]
[1,] "NY-T1"  "NY-T2"  "NY-T3"
[2,] "LA-T1"  "LA-T2"  "LA-T3"
[3,] "CHI-T1" "CHI-T2" "CHI-T3"
[4,] "HOU-T1" "HOU-T2" "HOU-T3"
```

将函数outer的第四个参数传递给paste。在这种情况下，传递sep="-"旨在定义一个连接字符串的分隔符。

outer的结果是一个矩阵。如果你需要生成的组合以向量形式出现，则调用函数as.vector来对它进行转换。

在特殊情况下，当你把一组字符串和它自身进行组合，而顺序并不重要时，结果将是重复的组合：

```
> outer(treatments, treatments, paste, sep="-")
     [,1]    [,2]    [,3]
[1,] "T1-T1" "T1-T2" "T1-T3"
[2,] "T2-T1" "T2-T2" "T2-T3"
[3,] "T3-T1" "T3-T2" "T3-T3"
```

但假设我们需要所有唯一的处理组合对。我们可以通过移除矩阵下三角（或者上三角）来排除重复的组合。函数lower.tri识别这个下三角，把它转换为所有在下三角之外的元素：

```
> m <- outer(treatments, treatments, paste, sep="-")
> m[!lower.tri(m)]
[1] "T1-T1" "T1-T2" "T2-T2" "T1-T3" "T2-T3" "T3-T3"
```

另请参阅

关于调用paste来生成字符串组合的内容，请参见方法7.2。

7.8 得到当前日期

问题

你需要知道今天的日期。

解决方案

函数Sys.Date返回当前日期：

```
> Sys.Date()
[1] "2010-02-11"
```

讨论

函数Sys.Date返回一个Date对象。在前面的例子中，它似乎返回一个字符串，因为结果输出在双引号内。然而，Sys.Date返回Date对象，R为了输出的目的把该日期对象转换为字符串。可以检查Sys.Date的结果来查看它的类：

```
> class(Sys.Date())
[1] "Date"
```

另请参阅

参见方法7.10。

7.9 转换字符串为日期

问题

你有一个表示日期的字符串，例如“2010-12-31”，而你想把它转换为一个Date对象。

解决方案

可以调用as.Date，但你必须知道字符串的格式。默认情况下，as.Date假定字符串看起来像*yyyy-mm-dd*。为了处理其他格式，你必须指定as.Date的参数format。例如，如果日期是美国风格的格式，则要设置参数format="%m/%d/%y"。

讨论

这个例子显示了as.Date假定的默认格式，它是ISO 8601标准格式*yyyy-mm-dd*：

```
> as.Date("2010-12-31")
[1] "2010-12-31"
```

函数as.Date返回一个Date对象，这里它为了输出被转换回字符串格式，这解释了输出结果两边的双引号。

该字符串可以是其他格式，但你必须提供一个参数format，使as.Date可以解释你的字符串。详情参见strftime函数的帮助说明页，了解有关允许的格式。

作为一个普通的美国人，我常常错误地尝试把美国的日期（*mm/dd/yyyy*）转换成Date对象，得到下列不愉快的结果：

```
> as.Date("12/31/2010")
Error in charToDate(x) :
    character string is not in a standard unambiguous format
```

以下是转换美国格式的日期的正确方法：

```
> as.Date("12/31/2010", format="%m/%d/%Y")
[1] "2010-12-31"
```

可观察到格式字符串Y被设置成大写以显示一个4位数字的年份。如果你使用2位数字的年份，则应该使用小写字体的y。

7.10 转换日期为字符串

问题

通常因为你需要输出一个日期，所以你需要把这个相应的Date对象转换为字符串。

解决方案

调用函数format或者as.character：

```
> format(Sys.Date())
[1] "2010-04-01"
> as.character(Sys.Date())
[1] "2010-04-01"
```

两个函数都允许用一个参数format来控制格式。例如，调用format="%m/%d/%Y"得到美国风格的日期：

```
> format(Sys.Date(), format="%m/%d/%Y")
[1] "04/01/2010"
```

讨论

参数format用来定义输出字符串的外观。标准字符（例如斜杠（/）或者连字符（-））都可以简单地复制到输出字符串中。百分号（%）和其后另外一个字母构成的两字母组合在这里具有特殊的意义。一些常见的组合有：

%b
: 缩写的月份名称（“Jan”）。

%B
: 完整的月份名称（“January”）。

%d
: 两位数字的日期。

%m
: 两位数字的月份。

%y
: 没有世纪的年份（00~99）。

%Y
: 有世纪的年份。

参见函数strftime的帮助说明页中有关格式化编码的完整列表。

7.11 转化年、月、日为日期

问题

你有代表一个日期的年份、月份以及日的元素，现在需要把该日期的这些元素合并成一个Date对象来表示。

解决方案

调用函数ISOdate：

```
> ISOdate(year, month, day)
```

结果是一个POSIXct对象，你可以把它转化为一个Date对象：

```
> as.Date(ISOdate(year, month, day))
```

讨论

一种常用的情况是，输入数据包含三个数字，它们分别表示日期的年、月和日。函数ISOdate可以把它们结合成一个POSIXct对象：

```
> ISOdate(2012,2,29)
[1] "2012-02-29 12:00:00 GMT"
```

在POSIXct格式中，你可以保持你的日期。然而，当处理纯日期（而不是日期和时间）时，我经常把它们转换为Date对象并且删掉不会使用到的时间信息：

```
> as.Date(ISOdate(2012,2,29))
[1] "2012-02-29"
```

尝试转换一个无效日期为NA：

```
> ISOdate(2013,2,29)          # Oops! 2013 is not a leap year
[1] NA
```

ISOdate可以处理整个年、月、日的向量，对于对输入数据的大批量转换，这是很方便的。下面的例子是开始于多个年份的1月的第三个星期三的年/月/日，把它们组合成Date对象：

```
> years
[1] 2010 2011 2012 2013 2014
> months
[1] 1 1 1 1 1
> days
[1] 15 21 20 18 17
> ISOdate(years, months, days)
[1] "2010-01-15 12:00:00 GMT" "2011-01-21 12:00:00 GMT"
[3] "2012-01-20 12:00:00 GMT" "2013-01-18 12:00:00 GMT"
[5] "2014-01-17 12:00:00 GMT"
> as.Date(ISOdate(years, months, days))
[1] "2010-01-15" "2011-01-21" "2012-01-20" "2013-01-18" "2014-01-17"
```

完美主义者会注意到月份向量是多余的，所以最后的表达式可以通过调用循环结构更加简化：

```
> as.Date(ISOdate(years, 1, days))
[1] "2010-01-15" "2011-01-21" "2012-01-20" "2013-01-18" "2014-01-17"
```

这个方法也可以通过调用函数ISOdatetime（详情参见帮助说明页），以拓展成处理年、月、日、小时、分钟和秒的数据：

```
> ISOdatetime(year, month, day, hour, minute, second)
```

7.12 得到儒略日期

问题

给定一个Date对象，你需要提取儒略日期（Julian date）。在R中，它是自1970年1月1日以来的天数。

解决方案

把Date对象转换为一个整数或者调用函数julian：

```
> d <- as.Date("2010-03-15")
> as.integer(d)
[1] 14683
> julian(d)
[1] 14683
attr(,"origin")
[1] "1970-01-01"
```

讨论

儒略"日期"是从某个任意起始点以来的天数。在R中，起始点是1970年1月1日，UNIX系统也是同一个起始点。所以1970年1月1日的Julian日期是零，如下所示。

```
> as.integer(as.Date("1970-01-01"))
[1] 0
> as.integer(as.Date("1970-01-02"))
[1] 1
> as.integer(as.Date("1970-01-03"))
[1] 2
.
. (etc.)
.
```

7.13 提取日期的一部分

问题

给定一个Date对象，你需要提取一部分日期，例如一周的某一天、一年的某一天、日历日、日历月或者日历年。

解决方案

把Date对象转换为一个POSIXlt对象，它是一个日期的各个部分的列表。然后从该列表中提取需要的部分：

```
> d <- as.Date("2010-03-15")
> p <- as.POSIXlt(d)
> p$mday                          # Day of the month
[1] 15
> p$mon                           # Month (0 = January)
[1] 2
> p$year + 1900                   # Year
[1] 2010
```

讨论

对象POSIXlt把一个日期表示为该日期的各个部分的列表。通过调用函数as.POSIXlt，把Date对象转换为POSIXlt对象，它会给出包含日期的这些相应部分的列表：

sec
: 秒数（0~61）。

min
: 分钟数（0~59）。

hour
: 小时数（0~23）。

mday
: 该月的天数（1~31）。

mon
: 月份（0~11）。

year
: 自1900起的年份。

wday
: 该周的某一天（0~6, 0=Sunday）。

yday
: 该年的某一天（0~365）。

isdst
: 夏时令标记。

调用这些日期部分，就可以知道2010年4月1日是星期四（wday = 4），是该年的第91天（因为在1月1日yday = 0）：

```
> d <- as.Date("2010-04-01")
> as.POSIXlt(d)$wday
[1] 4
```

```
> as.POSIXlt(d)$yday
[1] 90
```

一个常见的错误是忘记给返回的年份加上1900，该错误使你误以为你活在很久以前：

```
> as.POSIXlt(d)$year                    # Oops!
[1] 110
> as.POSIXlt(d)$year + 1900
[1] 2010
```

7.14 创建日期序列

问题

你想要创建一个日期序列，例如日子序列、月份或者年度日期。

解决方案

函数seq是一个泛型函数，它有一个Date对象的版本。与创建一个数字序列相似，它可以创建一个Date序列。

讨论

函数seq的典型用途是指定一个起始日期（from）、结束日期（to），以及增量（by）。增量1代表每天：

```
> s <- as.Date("2012-01-01")
> e <- as.Date("2012-02-01")
> seq(from=s, to=e, by=1)                # One month of dates
 [1] "2012-01-01" "2012-01-02" "2012-01-03" "2012-01-04" "2012-01-05" "2012-01-06"
 [7] "2012-01-07" "2012-01-08" "2012-01-09" "2012-01-10" "2012-01-11" "2012-01-12"
[13] "2012-01-13" "2012-01-14" "2012-01-15" "2012-01-16" "2012-01-17" "2012-01-18"
[19] "2012-01-19" "2012-01-20" "2012-01-21" "2012-01-22" "2012-01-23" "2012-01-24"
[25] "2012-01-25" "2012-01-26" "2012-01-27" "2012-01-28" "2012-01-29" "2012-01-30"
[31] "2012-01-31" "2012-02-01"
```

另一个典型用途是指定一个起始点日期（from）、增量（by），以及日期数（length.out）：

```
> seq(from=s, by=1, length.out=7)        # Dates, one week apart
[1] "2012-01-01" "2012-01-02" "2012-01-03" "2012-01-04" "2012-01-05" "2012-01-06"
[7] "2012-01-07"
```

增量（by）是灵活的，并且可以指定天、周、月或者年：

```
> seq(from=s, by="month", length.out=12)     # First of the month for one year
[1] "2012-01-01" "2012-02-01" "2012-03-01" "2012-04-01" "2012-05-01" "2012-06-01"
[7] "2012-07-01" "2012-08-01" "2012-09-01" "2012-10-01" "2012-11-01" "2012-12-01"
```

```
> seq(from=s, by="3 months", length.out=4)       # Quarterly dates for one year
[1] "2012-01-01" "2012-04-01" "2012-07-01" "2012-10-01"
> seq(from=s, by="year", length.out=10)          # Year-start dates for one decade
[1] "2012-01-01" "2013-01-01" "2014-01-01" "2015-01-01" "2016-01-01" "2017-01-01"
[7] "2018-01-01" "2019-01-01" "2020-01-01" "2021-01-01"
```

小心月份结尾部分的by="month"。在这个例子中，2月的结尾部分溢出到了3月，它可能不是你所要的结果：

```
> seq(as.Date("2010-01-29"), by="month", len=3)
[1] "2010-01-29" "2010-03-01" "2010-03-29"
```

第8章

概率

简介

概率论是统计学的基础，而R有许多用于处理概率、概率分布以及随机变量的机制。本章中的方法向你展示怎样计算分位数的概率，计算概率的分位数，生成给定分布的随机变量，绘制分布图等。

分布的名称

R对每个概率分布都有一个简称。这个名称用于识别与分布相联系的函数。例如，正态分布的名称是“norm”，它是这些函数名称的词根：

函数名	目的
dnorm	正态概率密度
pnorm	正态分布函数
qnorm	正态分位数函数
rnorm	正态分布的随机数

表8-1描述了一些常见的离散分布，表8-2描述了几个常见的连续分布。

表8-1：离散分布

离散分布名称	R函数	参数
二项分布	binom	n：试验次数；p：一次试验中事件成功的概率
几何分布	geom	p：一次试验中事件成功的概率

表8-1：离散分布（续）

离散分布名称	R函数	参数
超几何分布	hyper	*m*：瓮中白球个数；*n*：瓮中黑球个数；*k*：从瓮中抽取的球的个数
负二项分布	nbinom	size：成功的试验个数；或者prob：成功试验概率，或者mu：均值
泊松分布	pois	lambda：均值

表8-2：连续分布

连续分布名称	R函数	参数
贝塔分布	beta	shape1：形状1；shape2：形状2
柯西分布	cauchy	location：位置；scale：大小
卡方分布	chisq	df：自由度
指数分布	exp	rate：发生率
F分布	f	df1：第一自由度；df2：第二自由度
伽玛分布	gamma	rate：发生率；或者rate：发生率，或者scale：大小
对数正态分布	lnorm	meanlog：对数均值；sdlong:对数标准差
逻辑分布	logis	location：位置；scale：大小
正态分布	norm	mean：均值；sd：标准差
学生t分布	t	df：自由度
均匀分布	unif	min：左边界；max：右边界
威布尔分布	weibull	shap：形状；scale：大小
Wilcoxon分布	wilcox	*m*＝第一个样本的样本量 *n*＝第二个样本的样本量

警告： 所有与分布有关的函数都需要分布参数，例如二项分布的size和prob或者几何分布的prob。玄机在于分布参数也许不是你所期望的。例如，我希望一个指数分布的参数为β，它是均值。然而，R的惯例是把指数分布定义为rate = $1/\beta$，所以我经常提供错误的值。这个教训是，在调用一个与分布有关的函数前，查找帮助说明页。确定参数是正确的。

得到关于概率分布的帮助

为了了解R函数与一个特殊的概率分布有关，使用这个分布的帮助指令，并应用该分布的全称。例如，显示与正态分布相关的函数：

```
> ?Normal
```

有些分布的名称不容易通过help命令获取帮助信息，例如"学生*t*分布"。这些分布有特殊的帮助名称，如表8-1和表8-2所示：负二项分布（NegBinomial）、卡方分布（Chisquare）、对数正态分布（Lognormal）以及TDist。因此，为得到关于学生*t*分布的帮助，可用：

```
> ?TDist
```

另请参阅

有许多其他的分布在可下载的软件包中得以实现，参见概率分布的CRAN任务视图（*http://cran.r-project.org/web/views/Distributions.html*）。软件包SuppDists是R基础包的一部分，并且它包含10个补充分布。R软件包MASS也是基础包的一部分，它为分布提供附加支持，例如适用于一些普通分布的最大似然拟合、从多元正态分布中抽样等。

8.1 计算组合数

问题

你需要计算从*n*项中取*k*个的组合数目。

解决方案

调用choose函数：

```
> choose(n, k)
```

讨论

计算离散变量概率的一个普遍问题是计算组合：从有*n*个元素的集合中抽取大小为*k*的不同子集合的数目。该数目能够由$n!/r!(n-r)!$给出，但调用choose函数更为简便——尤其当*n*和*k*的值十分大时：

```
> choose(5,3)          # How many ways can we select 3 items from 5 items?
[1] 10
> choose(50,3)         # How many ways can we select 3 items from 50 items?
[1] 19600
> choose(50,30)        # How many ways can we select 30 items from 50 items?
[1] 4.712921e+13
```

这些数字称为二项式系数。

另请参阅

这个方法只是用来计算组合；参见方法8.2关于生成组合的内容。

8.2 生成组合

问题

你需要生成所有从*n*项中取*k*项的组合。

解决方案

调用combn函数：

```
> combn(items, k)
```

讨论

可以调用combn(1:5,3)来生成值为1~5，一次取3个数的所有组合：

```
> combn(1:5,3)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    1    1    1    1    1    2    2    2     3
[2,]    2    2    2    3    3    4    3    3    4     4
[3,]    3    4    5    4    5    5    4    5    5     5
```

这个函数不局限于数字。我们也可以生成字符串组合。下列是5项处理一次取3项的所有组合：

```
> combn(c("T1","T2","T3","T4","T5"), 3)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] "T1" "T1" "T1" "T1" "T1" "T1" "T2" "T2" "T2" "T3"
[2,] "T2" "T2" "T2" "T3" "T3" "T4" "T3" "T3" "T4" "T4"
[3,] "T3" "T4" "T5" "T4" "T5" "T5" "T4" "T5" "T5" "T5"
```

警告： 当项数*n*增加时，组合数会激增——尤其是，如果*k*偏离1或*n*时，较大时。

另请参阅

参见方法8.1，了解如何在生成一个大集合前，计算可能组合的数目。

8.3 生成随机数

问题

你需要生成随机数。

解决方案

生成一个0~1之间均匀分布的随机数可以由函数runif处理。下面的例子生成一个均匀随机数：

```
> runif(1)
```

然而，R能够从其他分布中生成随机变量。对于一个给定的分布，随机数生成程序的名称是放在分布的简称前的“r”（例如，rnorm是正态分布的随机数生成程序）。下面的例子从标准正态分布中生成一个随机值：

```
> rnorm(1)
```

讨论

大多数程序设计语言有一个功能薄弱的随机数生成程序，它只能生成一个随机数，在0.0~1.0之间均匀分布。R并非如此。

R可以从许多概率分布而不是仅仅从均匀分布中生成随机数。这个生成0~1之间的均匀随机数的简单情况可由runif函数处理：

```
> runif(1)
[1] 0.5119812
```

runif的参数是生成的随机数的数目。生成一个包含10个值的向量和生成一个值的向量同样容易：

```
> runif(10)
 [1] 0.03475948 0.88950680 0.90005434 0.95689496 0.55829493 0.18407604
 [7] 0.87814788 0.71057726 0.11140864 0.66392239
```

R实现的所有分布都有随机数生成程序。简单地在分布名称前加上“r”，你就拥有了相应的随机数生成程序的名称。下列是一些常见的随机数生成程序：

```
> runif(1, min=-3, max=3)          # One uniform variate between -3 and +3
[1] 2.954591
> rnorm(1)                         # One standard Normal variate
[1] 1.048491
```

```
> rnorm(1, mean=100, sd=15)          # One Normal variate, mean 100 and SD 15
[1] 108.7300
> rbinom(1, size=10, prob=0.5)       # One binomial variate
[1] 3
> rpois(1, lambda=10)                # One Poisson variate
[1] 13
> rexp(1, rate=0.1)                  # One exponential variate
[1] 8.430267
> rgamma(1, shape=2, rate=0.1)       # One gamma variate
[1] 20.47334
```

就runif而言，第一个参数是生成的随机数的数目。随后的参数是分布的参数，例如正态分布的mean和sd，或者二项分布的size和prob等。参见R函数的帮助说明页学习更多细节。

目前为止给出的例子对分布参数使用简单纯量。但参数也可以是向量，在这种情况下，R会在生成随机数时重复循环向量。下列例子生成了三个正态分布的随机数，这三个正态分布的均值分别为-10、0和+10，它们的标准偏差都是1：

```
> rnorm(3, mean=c(-10,0,+10), sd=1)
[1] -11.195667 2.615493 10.294831
```

在分层模型的情况下R有强大的性能，它的参数本身就是随机的。下面的例子生成100个正态随机数，它们的均值本身是超参数为$\mu = 0$和$\sigma = 0.2$的正态分布的随机数。

```
> means <- rnorm(100, mean=0, sd=0.2)
> rnorm(100, mean=means, sd=1)
  [1] -0.410299581 1.030662055 -0.920933054 -0.572994026  0.984743043
  [6]  1.251189879 0.873930251  0.509027943 -0.788626886 -0.113224062
 [11] -0.035042586 0.150925067 0.634036678 1.627473761 -0.812021925
 .
 . (etc.)
 .
```

如果你生成许多随机值并且参数向量很短，那么R会对参数向量应用循环规则。

另请参阅

参见本章的“简介”部分。

8.4 生成可再生的随机数

问题

你需要生成一个随机数序列，但你想要在你的程序每次运行时都复制这个同样的序列。

解决方案

在运行你的R代码之前，调用set.seed函数初始化随机数生成程序为一个已知状态：

```
> set.seed(666)       # Or use any other positive integer
```

讨论

在生成随机数之后，每次你的程序执行时，你可能需要复制这个同样的随机数序列。通过这种方式，你从反复运行中得到相同的结果。我曾经支持一个复杂的针对一个大型有价证券的蒙特卡罗分析。它的使用者抱怨每次程序运行都会得到一个稍微不同的结果。我回答：这都基于随机数，所以当然输出是随机的。解决方案是在程序的开始设置随机数生成程序为一个已知状态。这样的话，它会每次都生成相同（类似）的随机数，并且产生一致的、可复制的结果。

在R中，函数set.seed设置随机数生成程序为一个已知状态。该函数有一个整型参数。该参数取任何正整数都可以，但你每次必须使用同一个值以便得到一个相同的初始状态。

该函数没有返回值。它在幕后运行，初始化（或者重新初始化）随机数生成程序。这里的关键在于使用相同的种子重新启动随机数生成程序以回到相同的位置：

```
> set.seed(165)         # Initialize the random number generator to a known state
> runif(10)             # Generate ten random numbers
[1] 0.1159132 0.4498443 0.9955451 0.6106368 0.6159386 0.4261986 0.6664884
[8] 0.1680676 0.7878783 0.4421021
> set.seed(165)         # Reinitialize to the same known state
> runif(10)             # Generate the same ten "random" numbers
[1] 0.1159132 0.4498443 0.9955451 0.6106368 0.6159386 0.4261986 0.6664884
[8] 0.1680676 0.7878783 0.4421021
```

警告： 当你设置种子值并且冻结随机数序列时，你就排除了随机的一个来源，这对像蒙特卡罗模拟这样的算法至关重要。应用程序在调用set.seed之前，问问自己：我是否削弱了程序的功能，或许破坏了它的逻辑性？

另请参阅

参见方法8.3更多关于生成随机数的内容。

8.5 生成随机样本

问题

你需要对一个数据集随机取样。

解决方案

函数sample会随机地从一个向量中选定*n*项：

```
> sample(vec, n)
```

讨论

假设当世界职业大赛举办时，比赛数据包含一个年向量。你可以使用函数sample随机选定10年：

```
> sample(world.series$year, 10)
[1] 1906 1963 1966 1928 1905 1924 1961 1959 1927 1934
```

由于是随机选定的，所以再一次运行sample时，通常生成一个不同的结果：

```
> sample(world.series$year, 10)
[1] 1968 1947 1966 1916 1970 1961 1936 1913 1914 1958
```

函数sample是不放回抽样，这意味着它不会两次选定相同的项目。有些统计程序（尤其是自助法）需要放回抽样，这意味着一项可能会在一个样本中出现多次。对函数sample指定replace=TRUE为放回抽样。

使用放回抽样可以实施一个简单的自助抽样。以下代码片段重复从一个数据集x中进行抽样并且计算所有抽样样本的中位数：

```
medians <- numeric(1000)
for (i in 1:1000) {
    medians[i] <- median(sample(x, replace=TRUE))
}
```

从自助抽样的估计，我们可以估算出中位数的置信区间：

```
ci <- quantile(medians, c(0.025, 0.975))
cat("95% confidence interval is (", ci, ")\n")
```

典型的输出是：

```
95% confidence interval is ( 1.642021 1.702843 )
```

另请参阅

参见方法8.7关于随机排列一个向量和方法13.8更多关于自助法的内容。

8.6 生成随机序列

问题

你需要生成一个随机序列，例如一连串模拟抛硬币的结果或者一个伯努利试验的模拟序列。

解决方案

调用sample函数。从可能值集合中取样，并且设置replace=TRUE：

```
> sample(set, n, replace=TRUE)
```

讨论

函数sample从一个集合中随机选定项。它通常是不放回抽样，这意味着它不会两次选定相同的项。然而，设置replace=TRUE会使得样本可以反复选定项，这允许你生成长的随机项序列。

下面的例子生成一个抛10次硬币的模拟随机序列：

```
> sample(c("H","T"), 10, replace=TRUE)
 [1] "H" "H" "H" "T" "T" "H" "T" "H" "H" "T"
```

接下来的例子生成一个20次伯努利试验序列——随机地成功或者失败。用TRUE来表示一次成功：

```
> sample(c(FALSE,TRUE), 20, replace=TRUE)
[1] TRUE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE
[13] TRUE TRUE FALSE TRUE TRUE FALSE FALSE TRUE
```

样本会根据默认等概率地在集合元素中选择，所以选择TRUE或者FALSE的概率都为0.5。采用伯努利试验，成功的概率p不一定是0.5。你可以通过样本参数prob使抽样偏向某个事件，这个参数是一个概率向量，每个概率对应一个集合元素。假设我们需要生成20个伯努利试验，它成功的概率为$p=0.8$。我们设置FALSE的概率为0.2，TRUE的概率为0.8：

```
> sample(c(FALSE,TRUE), 20, replace=TRUE, prob=c(0.2,0.8))
 [1] TRUE TRUE TRUE TRUE TRUE TRUE  TRUE FALSE TRUE TRUE TRUE TRUE
[13] TRUE TRUE TRUE TRUE TRUE TRUE FALSE  TRUE
```

结果序列明显偏向TRUE。我选择这个例子，因为它是一个对一般技巧的简单演示。对于二元序列这个特殊例子，可以调用函数rbinom，二项式变量的随机生成程序为：

```
> rbinom(10, 1, 0.8)
[1] 1 0 1 1 1 1 1 1 1 1
```

8.7 随机排列向量

问题

你需要生成一个向量的随机排列。

解决方案

如果v是向量，那么sample(v)返回v的一个随机排列。

讨论

这里把函数sample用于大数据集的抽样。然而，该函数的默认参数使你能够生成一个数据集的随机重新排列。函数调用sample(v)等价于：

```
sample(v, size=length(v), replace=FALSE)
```

这意味着“以随机顺序选定v的所有元素并且每个元素只使用一次。这就是一个随机排列”。下列是一个1, ..., 10的随机排列：

```
> sample(1:10)
 [1]  5  8  7  4  3  9  2  6  1 10
```

另请参阅

参见方法8.5了解更多关于函数sample的内容。

8.8 计算离散分布的概率

问题

你需要计算样本或者与一个离散随机变量相关联的累积概率。

解决方案

对于一个简单概率$P(X = x)$，使用密度函数来计算。所有R中的概率分布都有一个密度

函数，其函数名称为前缀“d”加在分布名称前——例如，dbinom是二项分布的密度函数。

对于一个累积概率，$P(X \leqslant x)$，使用分布函数来计算。所有R中的概率分布都有一个分布函数，它的名称为前缀“p”加在分布名称前，因此，pbinom是二项分布的分布函数。

讨论

假设有一个二项随机变量X，总试验次数为10次，其中每个试验的成功概率为1/2。那么可以通过调用dbinom，计算观察到$x = 7$的概率：

```
> dbinom(7, size=10, prob=0.5)
[1] 0.1171875
```

以上代码计算的概率约为0.117。R调用密度函数dbinom。有些教科书中称它为概率质量函数（probability mass function）或者概率函数（probability function）。这里称它为密度函数，这样可以在离散分布和连续分布之间保持术语的一致性（参见方法8.9）。

累积概率$P(X \leqslant x)$由分布函数给出，它有时称为累积概率函数。二项分布的分布函数是pbinom。以下是$x = 7$的累积概率（例如，$P(X \leqslant 7)$）：

```
> pbinom(7, size=10, prob=0.5)
[1] 0.9453125
```

计算结果说明观察到$X \leqslant 7$的概率约为0.945。

下面是一些常用离散分布的密度函数和分布函数：

分布	密度函数：$P(X=x)$	分布函数：$P(X \leqslant x)$
二项分布	dbinom(x, size, prob)	pbinom(x, size, prob)
几何分布	dgeom(x, prob)	pgeom(x, prob)
泊松分布	dpois(x, lambda)	ppois(x, lambda)

累积概率的补集是生存函数，$P(X > x)$。所有的分布函数允许你通过指定lower.tail=FALSE简单地找到右尾概率，例如：

```
> pbinom(7, size=10, prob=0.5, lower.tail=FALSE)
[1] 0.0546875
```

因此我们可知观察到$X > 7$的概率约为0.055。

区间概率$P(x_1 < X \leqslant x_2)$是观察到X在范围x_1~x_2之间的概率。它可以作为两个累积概率之

间的差而简单计算出来：$P(X \leqslant x_2) - P(X \leqslant x_1)$。这里的$P(3 < X \leqslant 7)$对于我们的二项式变量为：

```
> pbinom(7,size=10,prob=0.5) - pbinom(3,size=10,prob=0.5)
[1] 0.7734375
```

R允许你用这些函数指定x的多个值并且返回一个相应的概率向量。这里我们计算两个累积概率，$P(X \leqslant 3)$和$P(X \leqslant 7)$，在其中调用pbinom：

```
> pbinom(c(3,7), size=10, prob=0.5)
[1] 0.1718750 0.9453125
```

这导致了计算区间概率的一个玄机。函数diff计算一个向量中连续两个元素的差。将它应用于pbinom的输出以获得累积概率的差——换句话说，区间概率为：

```
> diff(pbinom(c(3,7), size=10, prob=0.5))
[1] 0.7734375
```

另请参阅

参见本章“简介”了解更多有关R中概率分布的内容。

8.9 计算连续分布的概率

问题

你需要计算一个连续随机变量的分布函数（Distribution Function，DF）或者累积分布函数（Cumlative Distribution Function，CDF）。

解决方案

调用分布函数，它计算$P(X \leqslant x)$。所有R中的概率分布都有一个分布函数，它的名称为前缀“p”加在分布的简称前——例如，pnorm是正态分布的分布函数。

讨论

R中的概率分布函数遵循一个一致模式，所以本方法的解决方案本质上与离散随机变量的解决方案（参见方法8.8）一样。显著性差异是连续变量在单一点上没有“概率”，即$P(X = x)=0$。相反，它们在一点上有密度。

考虑到这个一致性，分布函数在方法8.8中的讨论在这里也可适用。下面的表格对多个连续分布给出分布函数：

分布	分布函数：$P(X \leqslant x)$
正态分布	pnorm(x, mean, sd)
学生t分布	pt(x, df)
指数分布	pexp(x, rate)
伽玛分布	pgamma(x, shape, rate)
卡方分布	pchisq(x, df)

我们可以调用pnorm来计算一个人矮于66英寸的概率，假设人的身高是一个均值为70英寸并且标准偏差为3英寸的正态分布。从数学上来说，我们需要在$X \sim N(70, 3)$的条件下计算$P(X \leqslant 66)$：

```
> pnorm(66, mean=70, sd=3)
[1] 0.09121122
```

同样地，我们可以调用pexp计算以40为均值的指数变量小于20的概率：

```
> pexp(20, rate=1/40)
[1] 0.3934693
```

与离散概率类似，对于连续概率的函数，可以设定参数lower.tail=FALSE来计算生存函数$P(X > x)$。下面对函数pexp的调用给出了指数变量大于50的概率：

```
> pexp(50, rate=1/40, lower.tail=FALSE)
[1] 0.2865048
```

同样，像离散概率一样，对于一个连续变量的区间概率$P(x_1 < X < x_2)$，可以作为两个累积概率的差计算，即$P(X < x_2) - P(X < x_1)$。对于上述指数变量，这里$P(20 < X < 50)$表示它可能落入20～50之间的概率：

```
> pexp(50,rate=1/40) - pexp(20,rate=1/40)
[1] 0.3200259
```

另请参阅

参见本章“简介”了解更多关于内置概率分布的内容。

8.10 转换概率为分位数

问题

给定一个概率p和一个分布，你需要决定相应p的分位数：即找到x值，使得$(X \leqslant x) = p$。

解决方案

R中的每个分布包含一个分位数函数，它转换概率为分位数。函数的名称为前缀“q”放在分布名称前，例如，qnorm是正态分布的分位数函数。

分位数函数的第一个参数是概率。剩下的参数是分布的参量，例如均值、形状，或者发生率：

```
> qnorm(0.05, mean=100, sd=15)
[1] 75.3272
```

讨论

计算分位数的一个常用例子是置信区间的计算。如果我们需要知道一个标准正态变量95%的置信区间（$\alpha = 0.05$），那么我们需要概率为$\alpha/2 = 0.025$和$(1 - \alpha)/2 = 0.975$的分位数：

```
> qnorm(0.025)
[1] -1.959964
> qnorm(0.975)
[1] 1.959964
```

按照R的设计准则，分位数函数的第一个参数可以是一个概率向量，这种情况下，我们得到一个分位数向量。我们可以用简单的一行来计算：

```
> qnorm(c(0.025,0.975))
[1] -1.959964 1.959964
```

所有的R内置概率分布提供一个分位数函数。下面是一些常用的离散分布的分位数函数：

分布	分位数函数
二项分布	qbinom(p, size, prob)
几何分布	qgeom(p, prob)
泊松分布	qpois(p, lambda)

下面是一些常用的连续分布的分位数函数：

分布	分位数函数
正态分布	qnorm(p, mean, sd)
学生*t*分布	qt(p, df)
指数分布	qexp(p, rate)

分布	分位数函数
伽玛分布	qgamma(p, shape, rate=*rate*)或qgamma(p, shape, scale=*scale*)
卡方分布	qchisq(p, df)

另请参阅

一个数据集的分位数不同于一个分布的分位数，详情参见方法9.5。

8.11 绘制密度函数

问题

你需要绘制一个概率分布的密度函数。

解决方案

在一定范围内定义一个向量*x*。将分布的密度函数应用于*x*并绘制它的结果。下列代码片段绘制了标准正态分布：

```
> x <- seq(from=-3, to=+3, length.out=100)
> plot(x, dnorm(x))
```

讨论

所有的R内置概率分布都包含一个密度函数。对于一个特殊的密度，函数名称是“d”前置到密度名称前。正态分布的密度函数是dnorm，伽马分布（gamma distribution）的密度为dgamma等。

如果密度函数的第一个参数是一个向量，那么函数在每一点计算它的密度并且返回密度的向量。

下面的代码创建了一个四个密度的2×2图，如图8-1所示。

```
x <- seq(from=0, to=6, length.out=100)          # Define the density domains
ylim <- c(0, 0.6)

par(mfrow=c(2,2))                               # Create a 2x2 plotting area

plot(x, dunif(x,min=2,max=4), main="Uniform",   # Plot a uniform density
        type='l', ylim=ylim)
plot(x, dnorm(x,mean=3,sd=1), main="Normal",    # Plot a Normal density
        type='l', ylim=ylim)
plot(x, dexp(x,rate=1/2), main="Exponential",   # Plot an exponential density
        type='l', ylim=ylim)
```

```
plot(x, dgamma(x,shape=2,rate=1), main="Gamma",  # Plot a gamma density
        type='l', ylim=ylim)
```

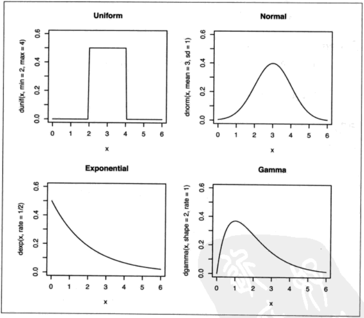


图8-1：绘制密度函数

原始的密度图很少有用或有趣，我们通常对感兴趣的区域打上阴影。图8-2显示一个在区域$1 \leqslant z \leqslant 2$打上阴影的标准正态分布。

我们通过首先绘制密度，然后调用polygon函数为一个区域打上阴影来创建图。函数polygon绘制一连串线围绕明亮区域并且填充它。

首先，我们画出密度曲线：

```
x <- seq(from=-3, to=+3, length.out=100)
y <- dnorm(x)
plot(x, y, main="Standard Normal Distribution", type='l',
     ylab="Density", xlab="Quantile")
abline(h=0)
```

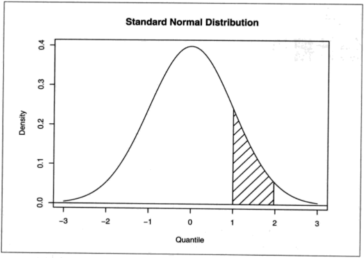


图8-2：有阴影的密度图

其次，我们用一系列线段定义感兴趣的区域。这些线段依次由一系列(x, y)坐标定义。函数polygon会连接第一个和最后一个(x, y)点来闭合这个多边形：

```
# The body of the polygon follows the density curve where 1 <= z <= 2
region.x <- x[1 <= x & x <= 2]
region.y <- y[1 <= x & x <= 2]

# We add initial and final  segments, which drop down to the Y axis
region.x <- c(region.x[1], region.x, tail(region.x,1))
region.y <- c(          0, region.y,                0)
```

最后，调用polygon来绘制区域的边界并且填充它：

```
polygon(region.x, region.y, density=10)
```

默认情况下，函数polygon不填充其内部区域。设置density=10将以倾斜45°的细线来填充内部区域。也可以用颜色来填充，可以通过设置density为-1并且设置col为需要的颜色：

```
polygon(region.x, region.y, density=-1, col="red")
```

第9章

统计概论

简介

R的任何有意义的应用都包括统计、模型或图形。本章讲述统计知识。有些方法简单地介绍了如何计算一个统计指标，如相对频率。大多数方法涉及统计检验或置信区间。统计检验让你选择两个相互矛盾的假设，具体过程在下面描述。置信区间反映总体参数的可能范围，它是基于样本数据来进行计算的。

原假设、备择假设和*p*值

本章中的许多统计检验都是久经考验的统计推断方法。在检验过程中，一般有一个或两个样本数据。两个相互矛盾的假设，它们中的一个是合理真实的。

其中的一个假设，称为原假设（null hypothesis），意为什么都没有发生：均值不变；治疗没有效果；你得到预期的答案；模型没有改善等。

另一个假设，称为备择假设（alternative hypothesis），意为某个事件发生：均值增加；治疗改善病人的健康；你得到了一个意料之外的答案；模型更好地适应等。

我们需要根据样本数据来判定哪个假设更有可能为真：

1. 首先，假设原假设为真。
2. 计算一个检验统计量。这可能是一个简单的指标，例如样本的均值，或者它也可能是相当复杂的指标。关键的要求是，我们必须知道统计量的分布。我们可能知道样本均值的分布，比如，通过应用中心极限定理。

3. 从统计量和它的分布中，我们可以计算出一个p值，它是在假设原假设为真时，得到最大的或者超出所得到的检验统计量值的概率。
4. 如果p值太小了，那么就有了证据反对原假设。这就是所谓的拒绝原假设。
5. 如果p值不小，那么就没有这样的证据。这就是所谓的不拒绝原假设。

这里有一个必需的判断：p值为多少是“太小”？

注意： 在本书中，我按照通常的惯例，当$p < 0.05$时，就拒绝原假设；当$p > 0.05$时，就不拒绝它。依照统计术语，选择显著水平$\alpha = 0.05$，作为反对原假设是否有强有力证据和证据不足之间的边界值。

然而，在实际中，“这需要依情况而定”。选择的显著水平取决于具体问题。惯例的极限$p < 0.05$适用于许多问题。在我的工作中，数据噪声较大，所以我常常取$p < 0.10$。有的是高风险领域，这时取$p < 0.01$或$p < 0.001$可能是必要的。

在这个方法中，我提到检验包括一个p值，这样你可以把p值与你所选择的显著性水平α相比较。本方法将帮助你解释这种比较。方法9.4（两个因素的独立检验）中提到：

> 按照惯例，p值小于0.05，表明变量可能不独立；然而p值超过0.05，则不能提供任何证据。

或者用下面的简洁说法：

- 原假设：变量是独立的。
- 备择假设：变量是不独立的。
- 对于$\alpha = 0.05$，如果$p < 0.05$，那么拒绝原假设，它给予了有力的证据证明变量是不独立的；如果$p > 0.05$，不能拒绝原假设。
- 你可以自由选择自己的α。当然，不同的α，拒绝或不拒绝的决定可能会有所不同。

记住：这里的方法给出的检验结果是非正式的解释，不是严格的数学解释。我使用通俗的语言是希望你有实际的理解，并了解检验的应用。如果假设检验的精确语义对你的工作是至关重要的，请参阅Wackerly等人的《Mathematical Statistic with Applications》第6版或者数理统计方面的其他优秀教科书。

置信区间

假设检验是一个已经被充分理解的数学过程，但它容易令人感到迷惑。首先，它的语义是很微妙的。检验并不能给出一个明确的、有用的结论。你可能会得到有力的证据来拒

绝原假设，但这就是你能得到的所有结论。其次，它不会给你一个数字，它给出的只有证据。

如果你想要得到数字，请使用置信区间，对一个给定的置信水平,它会给出一个总体参数的估计范围。本章中的方法可以计算均值、中位数以及总体比例的置信区间。

例如，方法9.9计算基于样本数据的95%的总体均值置信区间。它的区间是$97.16<\mu<103.98$，这意味着有95%的概率总体均值μ会落在97.16～103.98之间。

另请参阅

统计术语和惯例可以有所不同。本书一般遵循由Wackerly等人的《Mathematical statistic with Applications》，第6版（Duxbury出版社）的惯例。我推荐这本书以便了解更多本章描述的统计检验。

9.1 汇总数据

问题

你需要对数据有一个基本的统计汇总。

解决方案

函数summary对于向量、矩阵、因子和数据框给出了一些有用的统计汇总：

```
> summary(vec)
Min. 1st Qu. Median Mean 3rd Qu. Max.
0.1803 1.1090 1.9520 2.1750 2.6920 7.3290
```

讨论

解决方案展示了一个向量的汇总。1st Qu.和3rd Qu.分别是第一和第三分位数。还有中位数和均值，它们是十分有用的，你可以快速地察觉分布是否是偏斜的。例如，上例显示中位数要小于均值，这说明分布有可能向右偏斜。

对矩阵的汇总是逐列处理的。下面我们看到的是对一个矩阵数据mat的汇总，它的三个列分别为Samp1、Samp2、Samp3：

```
> summary(mat)
     Samp1              Samp2               Samp3
 Min.   :0.1803    Min.   : 0.1378    Min.   :0.1605
 1st Qu.:1.1094    1st Qu.: 1.2686    1st Qu.:1.0056
```

```
Median :1.9517   Median : 1.8733   Median :1.8466
Mean :2.1748     Mean : 2.2436     Mean :2.0562
3rd Qu.:2.6916   3rd Qu.: 2.9373   3rd Qu.:2.7666
Max. :7.3289     Max. :11.4175     Max. :6.3870
```

一个因子的汇总给出计数：

```
> summary(fac)
Yes No Maybe
44 20 36
```

数据框的汇总包含所有这些特征。它也是逐列处理，根据列的类型，给予适当的汇总。对数值型数据返回一个统计汇总，对因子数据返回计数（字符串未汇总）：

```
> summary(suburbs)
     city              county   state         pop
 Length:16          Cook    :7  IL:12   Min.   :  63348
 Class :character   Kane    :2  IN: 2   1st Qu.:  73833
 Mode  :character   Lake(IN) :2  WI: 2   Median :  86700
                    DuPage  :1          Mean   : 265042
                    Kendall :1          3rd Qu.: 103615
                    Kenosha :1          Max.   : 2853114
                    (Other) :2
```

一个列表的“汇总”是相当特别的：它仅给出每个列表元素的数据类型。这里是一个向量列表的汇总：

```
> summary(vec.list)
      Length Class Mode
Samp1 100 -none-  numeric
Samp2 100 -none-  numeric
Samp3 100 -none-  numeric
```

汇总一个向量列表，你需要对每个列表元素调用summary：

```
> lapply(vec.list, summary)
$Samp1
  Min. 1st Qu.  Median    Mean  3rd Qu.    Max.
0.1803  1.1090  1.9520 2.1750  2.6920  7.3290

$Samp2
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.1378  1.2690  1.8730 2.2440 2.9370  11.4200

$Samp3
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
0.1605  1.0060  1.8470 2.0560 2.7670   6.3870
```

不幸的是，summary函数不计算变量波动大小，例如标准差或绝对中位数偏差。这是一个严重的缺点，所以我在调用summary后，通常会马上调用函数sd或者函数mad。

另请参阅

参见方法2.6和方法6.2。

9.2 计算相对频数

问题

你需要计算样本中多个观测值的相对频数。

解决方案

通过使用一个逻辑表达式来识别你感兴趣的观测值，然后用mean函数来计算识别出的观测值的比例。例如，给定一个向量x，可以通过下列方法找到正数值的相对频数：

```
> mean(x > 0)
```

讨论

像$x>0$这样的逻辑表达式，产生一个逻辑值向量（TRUE和FALSE），每个值对应于某个元素x。函数mean将这些逻辑值分别转换成1和0，计算它们的均值。它给出了取值为TRUE的值的比例——即你感兴趣的观测值的相对频数。例如，在上面解决方案的例子中，这是正数值的相对频数。

这里的概念是比较简单的。最棘手的部分是设计一个合适的逻辑表达式。下面是一些例子：

`mean(lab == "NJ")`
: 取值为New Jersey的lab所占的比例。

`mean(after > before)`
: 加强了的观测值所占的比例。

`mean(abs(x-mean(x)) > 2*sd(x))`
: 偏离平均值超过两个标准差的观测值所占的比例。

`mean(diff(ts) > 0)`
: 在一个时间序列中，观测值大于它前一个观测值的观测所占的比例。

9.3 因子制表和列联表创建

问题

你需要把一个因子制成表格或者根据多个因子构建一个列联表。

解决方案

函数table产生一个因子的计数：

```
> table(f)
```

它也能从两个或者更多因子中产生列联表（交叉表）：

```
> table(f1, f2)
```

讨论

函数table计算一个因子的水平，例如计算因子initial和因子outcome的水平计数：

```
> table(initial)
initial
    Yes No Maybe
    37  36    27
> table(outcome)
outcome
Fail Pass
  47   53
```

函数table更强大的功能是产生列联表，也称为交叉表。列联表中的每个单元计算相应的“行-列组合”发生了多少次：

```
> table(initial, outcome)
       outcome
initial Fail Pass
  Yes     13   24
  No      24   12
  Maybe   10   17
```

这个表格显示了initial = Yes和outcome = Fail组合发生了13次，initial = Yes和outcome = Pass的组合发生了24次等。

另请参阅

函数xtabs也可以生成一个列联表。它有一个公式界面，有些人更喜欢使用这个函数。

9.4 检验分类变量独立性

问题

有两个因子类型的分类变量。需要应用卡方检验来验证它们是否相互独立。

解决方案

调用table函数从这两个因子生成一个列联表。然后应用summary函数来执行该列联表的卡方检验：

```
> summary(table(fac1,fac2))
```

输出结果包含一个*p*值。按照惯例，*p*值小于0.05，表示变量很可能不是独立的；而*p*值超过0.05，则不能提供变量不独立的任何证据。

讨论

这个例子对方法9.3中的列联表进行卡方检验，产生的*p*值为0.012 55：

```
> summary(table(initial,outcome))
Number of cases in table: 100
Number of factors: 2
Test for independence of all factors:
        Chisq = 8.757, df = 2, p-value = 0.01255
```

这里*p*值很小，它表明两个因子initial和outcome之间有可能不是独立的。从实际应用来说，它表明变量之间有一定的联系。

另请参阅

函数chisq.test也可以执行这个检验。

9.5 计算数据集的分位数（和四分位数）

问题

给定一个分数*f*，你想知道数据中相应该分数的分位数。也就是说，你寻求观测值*x*，使得小于*x*的观测值的比例为*f*。

解决方案

调用函数quantile。第二个参数是分数*f*：

```
> quantile(vec, f)
```

对于四分位数，可以省略第二个参数：

```
> quantile(vec)
```

讨论

假设vec包含1 000个0～1之间的观测值。函数quantile可以告诉你哪个观测值为数据的下5%边界：

```
> quantile(vec, .05)
        5%
0.04575003
```

如果把概率的含义理解为相对频率，那么quantile文档把该函数的第二个参数称为“概率”。

在真正的R风格中，第二个参数可以是一个概率向量。这种情况下，quantile返回一个相应分位数向量，每个分位数对应一个概率值：

```
> quantile(vec, c(.05, .95))
        5%        95%
0.04575003 0.95122306
```

上例是一个方便的识别数据中间的90%观测值的方法。

如果完全省略第二个参数，那么R假设你想要的概率为0、0.25、0.50、0.75和1.0，即四分位数：

```
> quantile(vec)
         0%         25%         50%         75%        100%
0.001285589 0.260075658 0.479866042 0.734801500 0.997817661
```

令人惊讶的是，函数quantile用九种不同的算法来实现分位数的计算。在你假设默认算法是最适合的算法之前，请参见函数的说明页。

9.6 求分位数的逆

问题

给定数据中一个观测值x，你想知道它相应的分数。也就是说，你想知道小于x的数据的比例。

解决方案

假设数据存储在一个向量vec中，把数据和给定观测值进行比较，然后使用mean计算小于*x*的观测值的相对频数：

```
> mean(vec < x)
```

讨论

表达式vec<x将vec向量的每个元素与x相比较，并返回一个逻辑值向量，其中如果vec[*n*]<x，则第*n*个逻辑值是TRUE。函数mean把这些逻辑值转换为0和1：0为FALSE，1为TRUE。所有1和0的均值是小于*x*的观测值的比例，或者称为*x*的逆分位数。

另请参阅

这是方法9.2中所描述方法的一个应用。

9.7 数据转换为*z*分数

问题

你有一个数据集，你需要计算所有数据元素的相应*z*分数（又称为规范化数据）。

解决方案

调用函数scale：

```
> scale(x)
```

这个方法用于向量、矩阵和数据框。在向量的情况下，scale返回规范化值向量。在矩阵和数据框的情况下，scale独立地规范化每一列并且在一个矩阵中返回规范化值的列。

讨论

你可能还需要对单一值*y*相对于数据集*x*进行规范化。这可以通过使用向量化运算进行，如下所示。

```
> (y - mean(x)) / sd(x)
```

9.8 检验样本均值（t检验）

问题

你从一个总体中得到一个样本。根据给定的这个样本，你想知道总体的均值是否相当于一个特殊值m。

解决方案

将函数t.test应用于样本x，使用参数mu=m：

```
> t.test(x, mu=m)
```

输出包括一个p值。按照惯例，如果$p<0.05$，则总体均值不太可能为m；而如果$p>0.05$则不能提供均值不等于m的证据。

如果样本量n是很小的，那么为了保证从t检验中获得的结果是有意义的，潜在的总体必须是正态分布的。多大的样本量为“小”呢？一个经验法则是指样本量$n<30$。

讨论

t检验是统计学的重要工具，下面是它的基本用途：从一个样本推论总体均值。下面的例子是从均值为$\mu=100$的正态总体来模拟抽样，然后采用t检验来检测总体均值是否可能是95，t.test反映的p值为0.001 897：

```
> x <- rnorm(50, mean=100, sd=15)
> t.test(x, mu=95)

        One Sample t-test

data: x
t = 3.2832, df = 49, p-value = 0.001897
alternative hypothesis: true mean is not equal to 95
95 percent confidence interval:
  97.16167 103.98297
sample estimates:
mean of x
 100.5723
```

p值很小，所以（根据样本数据）95不可能是总体均值。

通俗地说，我们可以解释p值低的原因。如果总体均值是95，那么得到检验统计量（$t=3.2832$或更极端的值）的概率只有0.001 897。这是不可能的，然而这是我们所观测到值。因此，我们得出这样的结论：原假设是错误的，所以样本数据不支持原假设：总体均值是95。

与此形成鲜明对比的是，对于均值为100的检验，给出的p值为0.737 4：

```
> t.test(x, mu=100)

        One Sample t-test

data:  x
t = 0.3372, df = 49, p-value = 0.7374
alternative hypothesis: true mean is not equal to 100
95 percent confidence interval:
  97.16167 103.98297
sample estimates:
mean of x
 100.5723
```

大的p值表明，样本数据与总体均值μ为100的假设是一致的。用统计术语来说，数据不提供证据拒绝均值为100的假设。

一个常见的情况是对均值为零进行检验。如果你省略了mu参数，它默认为零。

另请参阅

t.test是一个很奇妙的函数。参见方法9.9和方法9.15关于它的其他用途。

9.9 均值的置信区间

问题

对于一个取自总体的样本，需要计算总体均值的置信区间。

解决方案

将函数t.test应用于样本数据x：

```
> t.test(x)
```

它的输出包括95%置信水平的置信区间。为了得到其他置信水平的区间，调用参数conf.level。

正如方法9.8所描述的那样，如果样本量n较小，那么为了得到一个有意义的置信区间，总体必须是正态分布的。如前所述，判定样本量是否小的一个经验法则是$n<30$。

讨论

将函数t.test应用于一个向量将产生大量的输出。其中会有置信区间：

```
> x <- rnorm(50, mean=100, sd=15)
> t.test(x)

        One Sample t-test

data: x
t = 59.2578, df = 49, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
  97.16167 103.98297
sample estimates:
mean of x
 100.5723
```

在这个例子中，置信区间为约$97.16 < \mu < 103.98$，它有时简写为（97.16，103.98）。

通过设置conf.level=0.99，可以将置信水平提高到99%：

```
> t.test(x, conf.level=0.99)

        One Sample t-test

data: x
t = 59.2578, df = 49, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
99 percent confidence interval:
  96.0239 105.1207
sample estimates:
mean of x
 100.5723
```

以上改变把置信区间放宽为$96.02 < \mu < 105.12$。

9.10 中位数的置信区间

问题

你有一个数据样本，你想知道中位数的置信区间。

解决方案

调用函数wilcox.test，设置conf.int=TRUE：

```
> wilcox.test(x, conf.int=TRUE)
```

输出将包含这个中位数的置信区间。

讨论

计算均值的置信区间的过程是定义明确且广为人知的。不幸的是，同样的过程不适用于中位数。有多种计算中位数置信区间的过程。据我所知，它们都不是专门用于“计算中位数置信区间”的过程。应用Wilcoxon符号秩检验是一个非常标准的计算中位数置信区间的过程。

函数wilcox.test用于实现这个过程。输出中包括置信水平为95%的置信区间，在本例中大约是（0.424，0.892）：

```
> wilcox.test(x, conf.int=TRUE)

        Wilcoxon signed rank test

data: x
V = 465, p-value = 1.863e-09
alternative hypothesis: true location is not equal to 0
95 percent confidence interval:
 0.4235421 0.8922106
sample estimates:
(pseudo)median
     0.6249297
```

可以通过设置conf.level，例如设置成conf.level=0.99或者其他值来改变置信水平。

输出中也包括称为伪中位数的结果，它的定义可以在帮助说明页中查看。不要假定伪中位数等于中位数，它们是截然不同的：

```
> median(x)
[1] 0.547129
```

另请参阅

自助法对于估计中位数的置信区间也是很有用的；参见方法8.5和方法13.8。

9.11 检验样本比例

问题

现有取自总体的一个样本数据，它由成功与失败两个事件构成。你认为成功的比例是p，并且你需要使用样本数据来检验这个假设。

解决方案

调用函数prop.test。假设样本量是n并且样本包含x次成功：

```
> prop.test(x, n, p)
```

输出包括一个p值。依照惯例，p值小于0.05表示真实比例不太可能是p；然而p值大于0.05，则不能提供真实比例不等于p的证据。

讨论

假设你在棒球赛季开始时，遇到一些芝加哥小熊队（Chicago Cubs）吼叫着的球迷。队员打了20场比赛，并赢得了其中的11场，或者说，是赢得55%的比赛。基于这些证据，它的球迷“十分有信心”小熊队队员在今年会赢得超过一半的比赛。他应该如此确信吗？

函数prop.test可以评估球迷的逻辑。这里，观测值的数目是$n=20$，成功的次数是$x=11$，并且p是赢得比赛的真实概率。我们想知道基于这些数据，是否应该合理地得出结论：$P>0.5$。通常情况下，prop.test检验$P\neq0.05$的情况，但是我们可以通过设置参数alternative="greater"来检验$p>0.5$：

```
> prop.test(11, 20, 0.5, alternative="greater")

        1-sample proportions test with continuity correction

data: 11 out of 20, null probability 0.5
X-squared = 0.05, df = 1, p-value = 0.4115
alternative hypothesis: true p is greater than 0.5
95 percent confidence interval:
 0.3496150 1.0000000
sample estimates:
   p
0.55
```

函数prop.test的输出显示了一个大的p值，0.4115，所以我们不能拒绝原假设，即，我们不能合理地得出结论，p是大于1/2的。小熊队的球迷基于过少的数据，是对他们的球队过于自信了。这里是没有惊喜。

9.12 比例的置信区间

问题

你从总体中得到一个样本数据，它由成功与失败两种事件构成。基于这些样本数据，你需要给出总体成功比例的一个置信区间。

解决方案

调用函数prop.test。假设样本大小是n并且样本包含x次成功：

```
> prop.test(x, n)
```

这个函数的输出包括p的置信区间。

讨论

我订阅了一份关于股票市场的时事通讯，它内容的大部分都写得很好，但包括一个部分声称用来识别可能上涨的股票。它通过寻找股价的某个特定模式来进行上涨股票的识别。例如，它最近报道，某个股票遵循这个模式。它同样报告说，在最后模式发生的第9次后，股价又上涨了6次。因此，该文作者断定股价再次上涨的概率为6/9或66.7%。

调用prop.test，可以得到股价效仿这个模式上涨次数的真实比例的置信区间。这里，观测值的数目是$n = 9$并且成功的次数是$x = 6$。输出显示了在95%置信水平下置信区间为（0.309，0.910）：

```
> prop.test(6, 9)

        1-sample proportions test with continuity correction

data: 6 out of 9, null probability 0.5
X-squared = 0.4444, df = 1, p-value = 0.505
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.3091761 0.9095817
sample estimates:
        p
0.6666667
```

从上面的结果可知，声称“股价上涨的概率是66.7%”的文章的作者是很愚蠢的。他们可能将读者领入一个非常糟糕的赌注。

在默认情况下，prop.test计算在95%置信水平下的置信区间。应用参数conf.level 设定其他的置信水平：

```
> prop.test(x, n, p, conf.level=0.99)          # 99% confidence level
```

另请参阅

参见方法9.11。

9.13 检验正态性

问题

你需要一个统计检验来确定样本数据是否来自一个正态分布总体。

解决方案

调用函数shapiro.test：

```
> shapiro.test(x)
```

输出包括一个p值。依照惯例，$p<0.05$表示总体很可能不是正态分布，然而$p>0.05$没有提供这样的证据。

讨论

下面的例子显示了样本数据x的p值为0.4151：

```
> shapiro.test(x)

        Shapiro-Wilk normality test

data: x
W = 0.9651, p-value = 0.4151
```

较大的p值表明样本的总体可能是正态分布的。下一个例子显示样本数据y的p值较小，所以这个样本不可能来自一个正态总体：

```
> shapiro.test(y)

        Shapiro-Wilk normality test

data: y
W = 0.9503, p-value = 0.03520
```

这里的夏皮罗－威尔克（Shapiro–Wilk）检验是一个标准的R函数。你也可以安装R软件包nortest，它是专门用于正态检验的。这个软件包包括下列正态检验方法：

- Anderson-Darling检验（ad.test）
- Carmer-Von Mises检验（cvm.test）
- Lilliefors检验（lillie.test）
- Pearson卡方检验正态性复合假设（pearson.test）
- Shapiro-Francia检验（sf.test）

所有这些检验中存在的问题是它们的原假设：直到拒绝原假设前，它们都假设总体是正态分布的。因此，在检验显示一个较小p值前，总体必须是明显非正态的，然后就可以拒绝原假设。这使得检验显得非常保守，倾向于错误地过分证明正态性。

我建议同时使用直方图（参见方法10.18）和Q-Q图（参见方法10.21），而不是仅仅由统

计检验来判定样本数据的正态性。分布的尾部是否太扁平？或者图形峰值是否太高？相比单一的统计检验，从统计图得到的判断可能更好。

另请参阅

关于如何安装nortest软件包，请参阅方法3.9。

9.14 游程检验

问题

数据是只有两个取值的序列，例如“是”和“否”、“0”和“1”、“真”和“假”，或者其他两值的数据。你想知道：这个序列是随机的吗？

解决方案

软件包tseries包含函数runs.test，它检验一个序列是否是随机的。这个序列需要是一个有两个水平的因子：

```
> library(tseries)
> runs.test(as.factor(s))
```

函数runs.test显示一个*p*值。依照惯例，*p*值小于0.05，表示该序列可能不是随机的；然而*p*值大于0.05，不提供这样的证据。

讨论

游程（run）是一个由相同数值组成的序列，例如全部值为1或0。一个随机序列应该是这两个值的随机混合，没有太多的游程。同样，它也不应该包含过少的游程。一个完美的交替数值序列是（0，1，0，1，0，1，...），它不包含游程，但你会说它是随机的吗？

函数runs.test检验序列中的游程数。如果有过多或过少的游程，它便显示一个较小的*p*值。

第一个例子生成一个由0和1组成的随机序列，然后对该序列进行游程检验。果然，runs.test显示了一个较大的*p*值，表示序列很可能是随机的：

```
> library(tseries)
> s <- sample(c(0,1), 100, replace=T)
> runs.test(as.factor(s))

        Runs Test
```

```
data: as.factor(s)
Standard Normal = 0.2175, p-value = 0.8279
alternative hypothesis: two.sided
```

然而，下面的这个序列包含三个游程，所以显示的p值很小：

```
> s <- c(0,0,0,0,1,1,1,1,0,0,0,0)
> runs.test(as.factor(s))

        Runs Test

data: as.factor(s)
Standard Normal = -2.2997, p-value = 0.02147
alternative hypothesis: two.sided
```

另请参阅

参见方法5.4和方法8.6。

9.15 比较两个样本的均值

问题

你有分别来自两个总体的样本。你想知道这两个总体是否有相同的均值。

解决方案

通过调用函数t.test来执行t检验：

```
> t.test(x, y)
```

在默认情况下，t.test假设数据不是配对数据。如果观测值是配对的（即每个x_i与一个y_i配对），那么你需要指定参数paired=TRUE：

```
> t.test(x, y, paired=TRUE)
```

在任意一种情况下，t.test将计算出一个p值。依照惯例，如果$p < 0.05$，那么均值可能是不同的；然而$p > 0.05$，就不提供这样的证据：

- 如果其中一个样本量较小，那么总体必须为正态分布的。这里，“较小”是指少于20个数据点。
- 如果两个总体有相同的方差，指定参数var.equal=TRUE以获取较低的保守性检验（即更有效的检验）。

讨论

我经常使用t检验快速地检验两个总体之间是否存在差异。它要求样本量足够大（两个样本都要有20个或更多观测值）或者其所在的总体是正态分布的。我并不希望“正态分布”这个词让你望文生义。生成钟形的图形就足够好了。

这里的一个关键区别是数据是否包含成对的观测值，因为在这两种情况下的结果可能会有所不同。假设我们想知道在早晨喝咖啡是否能提高SAT的测试分数。我们可以用两种方式进行实验：

1. 随机选择一组人。让他们参加两次SAT考试，一次在早上喝咖啡，另一次早上不喝咖啡。对于每个人，我们将有两个SAT成绩。这就是成对观测值。
2. 随机选取两组人。一组在早晨喝咖啡并参加SAT考试。另一组不喝咖啡而参加考试。每个人有一个SAT成绩，但它们不以任何方式配对出现。

统计上显示，这些实验有很大不同。在实验1中，每个人有两个观测值（喝咖啡和不喝咖啡的）并且它们统计上是不独立的。在实验2中，数据是独立的。

如果你有成对观测值（实验1）并且错误地把它们作为未配对观测值（实验2）分析，那么你会得到结果的p值为0.986 7：

```
> t.test(x,y)

        Welch Two Sample t-test

data: x and y
t = -0.0166, df = 198, p-value = 0.9867
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -30.56737 30.05605
sample estimates:
mean of x mean of y
 501.2008 501.4565
```

较大的p值促使你得出结论：两组之间没有差异。然而，如果意识到数据是成对的，那么通过用配对方法分析相同数据，把检验结果和未配对的检验结果进行比较：

```
> t.test(x, y, paired=TRUE)

        Paired t-test

data: x and y
t = -2.3636, df = 99, p-value = 0.02005
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.4702824 -0.0410375
sample estimates:
```

```
mean of the differences
             -0.2556599
```

p值为0.020 05，得到了截然相反的结论。

另请参阅

如果总体不是正态分布（不是钟形图形）或者样本之一较小，那么考虑使用方法9.16中所描述的Wilcoxon–Mann–Whitney检验。

9.16 比较两个非参数样本的位置

问题

你有来自两个总体的样本。你不知道总体的分布，但你知道它们有类似的形状。你想知道：其中一个总体与另一个相比是否偏左或右？

解决方案

你可以使用非参数检验，即Wilcoxon-Mann-Whitney检验，它是由函数`wilcox.test`执行。对于成对观测值（即每个x_i与一个y_i配对），设置参数`paired=TRUE`：

```
> wilcox.test(x, y, paired=TRUE)
```

对于不配对的观测值，参数`paired`默认为`FALSE`：

```
> wilcox.test(x, y)
```

检验的输出包括一个p值。依照惯例，p值小于0.05，表示第二个总体对于第一个总体可能偏左或右；然而p值大于0.05，就不提供这样的证据。

讨论

当我们不再做出关于总体分布的假设时，我们就进入了非参数统计的世界。t检验要求数据是正态分布的（对于小样本），而Wilcoxon-Mann-Whitney检验是非参数的，因此它比t检验适用于更多的数据集。该检验的唯一的假设是：两个总体具有相同的图形性状。

在这个方法中，我们提出这样的问题：第二个总体相比第一个是偏左或偏右了吗？这相当于在问：第二个总体的均值是否小于或大于第一个总体的均值。然而，Wilcoxon-Mann-Whitney检验回答了一个不同的问题：它告诉我们两个总体的中央位置是否有显著的不同，或者等同于，它们的相对频率是否不同。

假设我们随机地选择一组员工，并要求每个人在两种不同的情况下完成相同的任务：在有利的情况下和不利的情况下，例如在嘈杂的环境中。我们衡量他们在两种情况下完成任务的时间，所以我们对每一位员工有两个测量值。我们想知道这两个时间是否有显著不同，但我们不能假设它们是正态分布的。

由于数据是成对的，所以我们必须设置paired=TRUE：

```
> wilcox.test(fav, unfav, paired=TRUE)

        Wilcoxon signed rank test with continuity correction

data: fav and unfav
V = 0, p-value < 2.2e-16
alternative hypothesis: true location shift is not equal to 0
```

p值近似等于零。从统计学角度来说，我们拒绝完成时间都是相同的假设。实际上来讲，完成时间不同的结论是合理的。

在这个例子中，设置参数paired=TRUE是至关重要的。由于观测值之间是不独立的，所以不能把数据看做是不配对的，否则会产生错误的结果。在设置paired=FALSE的情况下，运行这个例子产生的p值为0.229 8，从而导致错误的结论。

另请参阅

参见方法9.15关于参数检验的内容。

9.17 检验相关系数的显著性

问题

你在计算两个变量的相关系数，但你不知道它在统计意义上是否显著。

解决方案

函数cor.test可以计算出相关系数的p值和置信区间。如果变量来自正态分布的总体，那么使用相关系数的默认算法，即Pearson法：

```
> cor.test(x, y)
```

对于非正态总体，使用Spearman法：

```
> cor.test(x, y, method="Spearman")
```

该函数返回多个值，包括检验显著性得到的*p*值。依照惯例，$p < 0.05$表明相关系数可能是显著的，然而$p > 0.05$表明它不显著。

讨论

在我的经验中，人们往往不对相关系数的显著性进行检验。事实上，许多人不知道相关系数可以是不显著的。他们把数据输入计算机，计算相关系数，并且盲目地相信结果。然而，他们应该问问自己：有足够的数据吗？相关系数足够大吗？幸运的是，函数cor.test可以回答这些问题。

假设有两个向量*x*和*y*，它们都取自正态总体。我们可能满足于它们的相关系数大于0.83：

```
> cor(x, y)
[1] 0.8352458
```

但这种想法是很天真的。如果我们运行cor.test，它显示一个较大的*p*值，0.164 8：

```
> cor.test(x, y)

        Pearson's product-moment correlation

data: x and y
t = 2.1481, df = 2, p-value = 0.1648
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.6379590 0.9964437
sample estimates:
      cor
0.8352458
```

*p*值大于惯例定义的阈值0.05，所以我们认为相关系数不太可能是显著的。

你也可以通过使用置信区间来检验相关系数。在这个例子中，置信区间为（−0.638，0.996）。区间包含零，因此有可能相关系数为零，在这种情况下就没有相关性。因此，你不能确信你的相关系数是显著的。

cor.test的输出还包括函数cor能输出的相关系数的点估计（在底端，标识有“样本估计”），这为你节省运行cor的额外步骤。

默认情况下，cor.test计算Pearson相关分析，它假设潜在的总体是正态分布的。Spearman法没有做出这样的假设，因为它是非参数估计。当处理非正态数据时，使用method="Spearman"。

另请参阅

关于计算简单相关的内容，参见方法2.6。

9.18 检验组的等比例

问题

你有从两个或多个组中得到的样本。该组的元素是二值数据：成功或者失败。你想知道这些组是否有相等的成功比例。

解决方案

调用函数prop.test，它有两个向量参数：

```
> ns <- c(ns₁, ns₂, ..., nsₙ)
> nt <- c(nt₁, nt₂, ..., ntₙ)
> prop.test(ns, nt)
```

两个参数是平行向量。第一个向量ns给出了每一组的成功次数。第二个变量nt给出了相应组的大小（通常称为试验次数）。

输出包括一个p值。依照惯例，p值小于0.05表示组的比例可能不同，然而p值超过0.05不提供这样的证据。

讨论

在方法9.11中，我们讨论了基于一个样本的比例检验。这里，我们有取自多个总体的样本，需要比较它们所在总体的比例。

我最近教38名学生统计学，其中14人获得了A级成绩。我的一名同事教同样的学科，有40名学生，其中只有10人得到了A。我想要知道的是，与她相比，我是否显著地比她更容易给学生A级成绩而造成分数贬值？

我调用prop.test。“成功”是指颁发的成绩为A，所以成功向量包含两个元素：由我给的A数量和由我的同事给的A数量：

```
> successes <- c(14,10)
```

实验次数是相应班级的学生数：

```
> trials <- c(38,40)
```

函数prop.test的输出产生一个p值为0.374 9：

```
> prop.test(successes, trials)

        2-sample test for equality of proportions with continuity correction

data: successes out of trials
X-squared = 0.7872, df = 1, p-value = 0.3749
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.1110245 0.3478666
sample estimates:
   prop 1    prop 2
0.3684211 0.2500000
```

较大的p值意味着我们不能拒绝原假设，即没有证据说明我的评级和她的评级之间存在显著差异。

另请参阅

参见方法9.11。

9.19 组均值间成对比较

问题

你有多个样本，要在样本均值之间执行成对比较。也就是说，你想把一些样本的均值与其他任何一个样本的均值进行比较。

解决方案

将所有数据放入一个向量并创建一个平行因子来识别数据的组别。调用函数pairwise.t.test执行均值的成对比较：

```
> pairwise.t.test(x,f)          # x contains the data, f is the grouping factor
```

输出包含一个p值的表格，其中每个p值对应每一组对。依照惯例，如果$p<0.05$，那么这两组可能有不同的均值；然而$p>0.05$，没有提供这样的证据。

讨论

这方法比方法9.15更加复杂，方法9.15仅仅比较了两个样本的均值。这里，有多个样品，并要把任何一个样本的均值与其他任何一个样本的均值进行比较。

从统计学上来讲，成对比较很棘手。这与简单地对每一个可能的对子执行t检验并不相同。p值必须调整，否则你会得到一个过于乐观的结果。函数pairwise.t.test和函数p.adjust的帮助说明页描述了R中提供的调整算法。建议任何需要进行严肃成对比较的人参考这些函数的帮助说明页面并查阅相关主题的书籍。

假设我们使用方法5.5中的数据，在5.5节中，我们把大学新生、大二学生以及大三学生的数据组合到一个名为comb的数据框中。该数据框有两列：一列数据称为values，另一列中的分组因子称为ind。我们可以调用pairwise.t.test来执行组之间的成对比较：

```
> pairwise.t.test(comb$values, comb$ind)

        Pairwise comparisons using t tests with pooled SD

data: comb$values and comb$ind

     fresh jrs
jrs 0.0043 -
soph 0.0193 0.1621

P value adjustment method: holm
```

注意，这个p值的表格，大三（jrs）与大一（fresh）的比较，以及大二（soph）与大一（fresh）的比较都产生了一个较小的p值：分别为0.0043和0.0193。我们可以断定这些组之间有显著性差异。然而，大二（soph）与大三学生（jrs）的比较产生一个（相对）较大的p值，0.1621，所以它们没有显著差异。

另请参阅

参见方法5.5和方法9.15。

9.20 检验两样本的相同分布

问题

你有两个样本，你的问题是：它们取自同一个分布吗？

解决方案

Kolmogorov-Smirnov检验比较两个样本并检验它们是否取自于相同的分布。函数ks.test实现了这个检验：

```
> ks.test(x, y)
```

输出包括一个p值。依照惯例，p值小于0.05，表示两个样本（x和y）来自不同的分布；然而p值大于0.05，没有提供这样的证据。

讨论

Kolmogorov-Smirnov检验的强大优势在于两点。首先，它是一个非参数检验，所以不需要做任何关于数据分布的假设：它适用于所有分布。其次，它基于样本数据来检验总体的位置、离差和形状。如果这些特征不一致，那么该检验会检测到，并允许我们对基础分布做出不同的结论。

假设我们怀疑向量x和y来自不同的分布。这里，ks.test显示p值为0.012 97：

```
> ks.test(x, y)

        Two-sample Kolmogorov-Smirnov test

data: x and y
D = 0.3333, p-value = 0.01297
alternative hypothesis: two-sided
```

从较小的p值，我们可以得出结论：该样本服从不同的分布。然而，当我们对x和另一个样本z进行检验时，p值大很多（0.8245），这表明x和z可以有相同的总体分布。

```
> ks.test(x, z)

        Two-sample Kolmogorov-Smirnov test

data: x and z
D = 0.1333, p-value = 0.8245
alternative hypothesis: two-sided
```

第10章

图形

简介

图形是R的一个强大功能。软件包graphics是R标准发布版的一部分，它包含许多有用的函数以创建各种图形显示。本章重点介绍这些函数，偶尔会提到其他软件包。在本章的“另请参阅”部分提到的其他软件包中的函数，它们以不同的方式完成同样的处理。如果你不满足于基本函数，我建议你探索这些替代选择。

图形是一个广泛的课题，这里只能做肤浅的论述。如果你想深入钻研，我推荐Paul Murrell的著作《R Graphics》（Chapman & Hall，2006）。这本书讨论R图形背后的范例，说明如何使用图形函数，并包含许多例子——其中包括创建书中图形的代码。书中的一些例子是相当的令人惊叹。

插图

本章的图形大多平淡而朴素。这是我有意为之的。当调用plot函数时：

```
> plot(x)
```

得到x一个简单的图形表示。可以用颜色、标题、标签、图例和文本等来装饰图形，但这样会使函数plot的调用变得越来越复杂，模糊了原始的基本意图，例如：

```
> plot(x, main="Forecast Results", xlab="Month", ylab="Production",
+      col=c("red", "black", "green"))
```

我想使本章的方法比较清晰，所以我强调基本绘图，然后在本书后面的部分（例如方法10.2）再论述如何添加装饰。

图形函数的注意事项

了解高级图形函数和低级图形函数之间的区别是十分重要的。高级图形函数启动一个新的图形。它初始化图形窗口（在有必要时创建它）、设置标度，也许会绘制一些装饰，例如标题和标签，最后呈现图形。例子包括：

`plot`
: 通用绘图函数。

`boxplot`
: 创建箱线图。

`hist`
: 创建直方图。

`qqnorm`
: 创建Q-Q图。

`curve`
: 绘制函数图形。

低级的图形函数不能启动新图形。相反，它对已存在的图形添加装饰：例如点、线、文本、装饰等。例子包括：

`points`
: 添加点。

`lines`
: 添加线。

`abline`
: 添加直线。

`segments`
: 添加线段。

`polygon`
: 添加闭合多边形。

`text`
: 添加文本。

在调用一个低级图形例程之前，必须先调用一个高级图形例程。低级例程需要已初始化的图形；否则，将得到一个错误，如下所示。

```
> abline(a=0, b=1)
Error in int_abline(a = a, b = b, h = h, v = v, untf = untf, ...) :
    plot.new has not been called yet
```

通用绘图函数

在本章中，我们会面对R的多态性。多态函数或泛型函数根据输入参数的类型来决定处理的方式。函数plot是多态的。根据x是向量、因子、数据框、线性回归模型、表格或其他类型，函数plot(x)会产生不同的结果。

在本章的方法中，你会看到plot得到反复使用。每次使用它，仔细注意参数的类型。这将有助于你理解和记住为何函数如此运转。

其他软件包中的图形

R是高度可编程的，并且很多人用附加特性来扩充图形机制。很多时候，R软件包包括专门来绘制该包的结果和对象的函数。例如，软件包zoo实现时间序列对象。如果创建一个zoo的对象z，并且调用plot(z)，那么R软件包zoo将进行图形绘制，它创建一个专门用于显示时间序列的图形。

甚至有整个软件包致力于扩充R的新图形范例。软件包lattice是传统图形一个替代选项。它采用了更强大的图形范例，使你能够更轻松地创建包含更多信息的图形。它的结果一般也更好阅读。它由Deepayan Sarkar开发，作者同时也撰写了《Lattice：Multivariate Data Visualization with R》(Springer，2008)一书，这本书解释了lattice软件包以及如何使用它。软件包lattice也在《R in a Nutshell》（O'Reilly）中有所描述。

ggplot2软件包提供了另一个图形范例，这就是所谓的图形的语法。它对图形采用了高度模块化的方法，它让你更容易地构建和定制图形。这些图形一般也都比传统的图形更具吸引力。ggplot2软件包由Hadley Wickham创建，他还写了《ggplot2：Elegant Graphics for Data Analysis》（Springer，2009）一书。该书介绍了ggplot2背后的范例和如何使用该软件包的函数。

10.1 创建散点图

问题

你有成对的观测值：$(x_1, y_1)(x_2, y_2),\ldots,(x_n, y_n)$。你想创建一个成对数据的散点图。

解决方案

如果数据在两个平行向量*x*和*y*中，那么把它们当做plot的参数使用：

```
> plot(x, y)
```

如果数据在一个（两列）数据框中，绘制数据框如下：

```
> plot(dfrm)
```

讨论

散点图通常是处理新数据集的第一步。如果在*x*和*y*之间有任何关系，它是一个看清数据间关系的快捷方法。创建散点图很简单：

```
> plot(x, y)
```

函数plot不返回任何结果。相反，它的目的是在图形窗口绘制（*x*, *y*）数据对的图形。

如果数据存储在一个两列的数据框中，那么处理它将更加容易。如果用plot处理一个两列的数据框，该函数假设你需要一个根据两列创建的散点图。图10-1所示的散点图由调用plot创建：

```
> plot(cars)
```

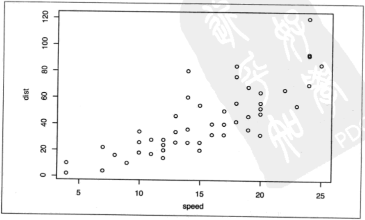


图10-1：散点图

数据集cars包含两列，speed和dist。第一列是speed，所以它作为x轴而dist作为y轴。

如果数据框包含两个以上的列，那么会得到多个散点图，这可能没有用（参见方法10.7）。

为了得到散点图，数据必须是数值型的。前面讲过plot是一个多态函数，所以如果参数为非数值型，那么它会绘制出一些其他图形。例如，参见方法10.16，它创建了因子的箱线图。

另请参阅

关于添加标题和标签的内容，参见方法10.2；关于分别添加网格和图例的内容，参见方法10.3和方法10.5；关于绘制多个变量的内容，参见方法10.7。

10.2 添加标题和标签

问题

需要给图形添加一个标题或给坐标轴添加多个标签。

解决方案

当调用plot时：

- 使用参数main添加一个标题。
- 使用参数xlab添加一个x轴标签。
- 使用参数ylab添加一个y轴标签。

```
> plot(x, main="The Title", xlab="X-axis Label", ylab="Y-axis Label")
```

另外一种方式是，在绘制数据时设置参数ann=FALSE，它要求不绘制注释内容，然后调用函数title添加需要的标题和标签，例如：

```
> plot(x, ann=FALSE)
> title(main="The Title", xlab="X Axis Label", ylab="Y Axis Label")
```

讨论

方法10.1中的图形是很普通的。它需要一个标题和多个更好的坐标轴标签，这样图形看起来更加有趣并且更容易解释。可以通过plot的参数来添加它们，产生的图形如图10-2所示。

```
> plot(cars,
+      main="cars: Speed vs. Stopping Distance (1920)",
+      xlab="Speed (MPH)",
+      ylab="Stopping Distance (ft)")
```

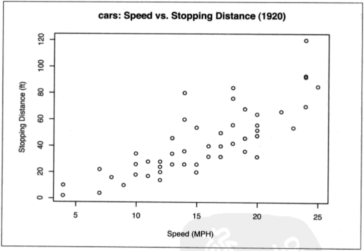


图10-2：添加一个标题和多个标签

现在默认图形有了明显的改善。其中，标题告诉我们一些关于图形的信息，而标签给出了计量单位。

10.3 添加网格

问题

需要对图形添加网格。

解决方案

- 调用函数plot时设置参数type="n"，它将在不显示数据的情况下初始化图形框架。
- 调用函数grid来绘制网格。

- 调用低级图形函数，例如points和lines以绘制覆盖在网格上的图形。

```
> plot(x, y, type="n")
> grid()
> points(x, y)
```

讨论

函数plot不会自动绘制网格。然而，对于某些图形，网格对读者是有帮助的。

以上解决方案显示了添加一个网格的适当方法。在绘制图形之前，网格必须先绘制。如果先画图形，网格将覆盖图形或者部分隐藏它们。

可以对方法10.2的图形添加网格。首先，用参数type="n"绘制图形。这将创建一个新的图形，包括标题和坐标轴，但没有（或尚未）绘制出数据：

```
> plot(cars,
+      main="cars: Speed vs. Stopping Distance (1920)",
+      xlab="Speed (MPH)",
+      ylab="Stopping Distance (ft)",
+      type="n")
```

然后，绘制网格：

```
> grid()
```

最后，通过调用低级图形函数points来绘制散点图，最终创建的图形如图10-3所示。

```
> points(cars)
```

如果急于处理数据，那么这里有一个绘制带有网格的图形的捷径：可以像通常一样调用函数plot，然后调用函数grid。唯一的问题是网格会覆盖图形，或者部分覆盖图形。有时候，它会产生明显的问题，而有时不会。你可以尝试一下。

10.4 创建多组散点图

问题

在两个向量*x*和*y*中有成对观测值，还有一个平行因子*f*，它表明观测值的组别。你想创建一个可以区分不同组别的*x*和*y*的散点图。

解决方案

使用plot的参数pch。它根据组，使用不同的绘图字符来绘制每个点：

```
> plot(x, y, pch=as.integer(f))
```

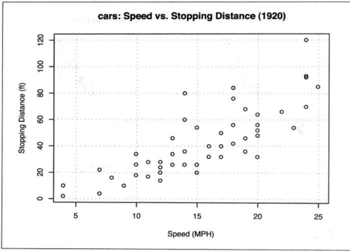


图10-3：添加网格

讨论

除非有办法将一组区别于另一组，否则在一个散点图中绘制多个组会产生一个信息空洞的混乱数据图形。函数plot的参数pch让我们对每个(x, y)数据对使用不同的绘图字符来绘制点。

数据集iris包含配对的量度Petal.Length和Petal.Width。每个量度也有一个Species属性表明已衡量的花的种类。如果一次性绘制所有的数据，就会得到显示在图10-4a中的散点图：

```
> with(iris, plot(Petal.Length, Petal.Width))
```

如果根据花的种类区别数据点，则图形将提供更多的信息。参数pch是一个整数向量，每个整数相对于一个(x, y)点，它们的值介于0～18之间。数据点将根据它们在pch中相应的值来绘制。这种调用plot的方式是根据组来绘制每个点的：

```
> with(iris, plot(Petal.Length, Petal.Width, pch=as.integer(Species)))
```

结果显示在图10-4b中，它清楚地表明根据数据在组中的种类绘制点。

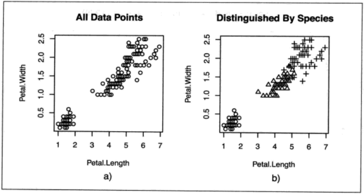


图10-4：多组散点图

另请参阅

关于添加说明的内容，请参见方法10.5。

10.5 添加图例

问题

图形需要包含一个图例，即添加小方框为读者解释图形中的元素。

解决方案

调用plot后，调用函数legend。前三个参数总是相同的，但之后的参数会作为标签变化的函数。以下是如何创建点、线和颜色的图例：

点的图例

```
legend(x, y, labels, pch=c(pointtype1, pointtype2, ...))
```

不同线型的线的图例

```
legend(x, y, labels, lty=c(linetype1, linetype2, ...))
```

不同线宽的线的图例

```
legend(x, y, labels, lwd=c(width1, width2, ...))
```

颜色的图例

```
legend(x, y, labels, col=c(color1, color2, ...))
```

这里，x和y是图例框的坐标，而labels是出现在图例中的字符串向量。参数pch、lty、lwd和col是平行于参数labels的向量。每个标签将显示在它所对应的点类型、线型、线宽，或颜色的旁边，具体设置由你指定哪一个来决定。

讨论

图10-5a显示了一个图例，它添加至方法10.4的散点图中。图例告诉读者哪些点与哪些种类相关。可以用以下方法添加图例（在调用plot之后）：

```
> legend(1.5, 2.4, c("setosa","versicolor","virginica"), pch=1:3)
```

前两个参数给出了图例框的x和y坐标。第三个参数是一个标签向量。在本例中，它是花的种类名称。最后一个参数表示根据点类型来标记点——具体而言，类型1～3。

事实上，我用以下方法创建了上面的图形和图例：

```
> f <- factor(iris$Species)
> with(iris, plot(Petal.Length, Petal.Width, pch=as.integer(f)))
> legend(1.5, 2.4, as.character(levels(f)), pch=1:length(levels(f)))
```

这里的奇妙之处在于花的种类名称不是硬编码到R代码中的。相反，一切是由种类因子f驱动的。我喜欢这种安排的原因有两个：首先，它意味着我不会忘记或拼错花的种类名称。其次，如果有人增加了其他的种类，代码会自动适应这个增加。第二个优势，在R脚本中尤其有价值。

图10-5b的图形显示了根据线型（实线、虚线或点线）的线图例。创建它的代码如下：

```
> legend(0.5, 95, c("Estimate","Lower conf lim","Upper conf lim"),
+        lty=c("solid","dashed","dotted"))
```

这个方法中的两个例子展现了点类型和线型的图例。函数legend也可以为颜色、填充内容和线宽创建图例。该函数还有许多参数和选项，它们用来控制图例的精确外观。详情参见帮助页。

通常情况下，图例有一个奇妙的功能，但函数legend会造成两个麻烦。首先，你必须自己选择图例的x和y坐标。R不能为你挑选图例坐标。可以先绘制没有图例的数据图形，然后挑一个空白区域把图例放上去；否则，图例可能会覆盖图形。

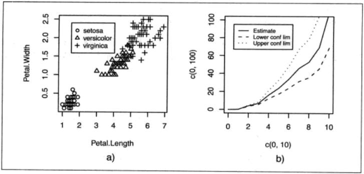


图10-5：图例的例子

其次，标签和点类型（或线型或颜色）的对应不是自动生成的。你有责任保证它们正确地对应。这个任务很容易出现错误并且不容易进行自动化。

10.6 绘制散点图的回归线

问题

绘制数据点对，同时需要添加一条说明它们线性回归关系的线。

解决方案

创建模型对象，绘制 (x, y) 对，然后调用abline函数绘制模型对象：

```
> m <- lm(y ~ x)
> plot(y ~ x)
> abline(m)
```

讨论

当使用abline(m)绘制模型对象时，它绘制的是最佳拟合回归线。这与plot(m)函数不同。plot函数生成的是回归分析的诊断图（参见方法11.15）。

假设用R软件包faraway中的数据集strongx来建模：

```
> library(faraway)
> data(strongx)
> m <- lm(crossx ~ energy, data=strongx)
```

可以在散点图中显示数据，然后调用abline添加回归线，生成图10-6：

```
> plot(crossx ~ energy, data=strongx)
> abline(m)
```

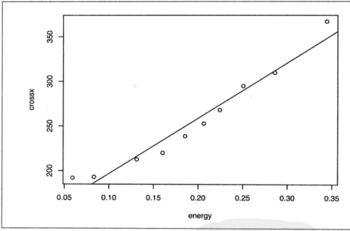


图10-6：包含回归线的散点图

注意，用一个公式来定义绘图，（即crossx ~ energy）。我喜欢使用这个语法，因为通过它可以很快看到拟合的回归直线与回归公式是对等的。另一种方法是调用plot(strongx$energy, strongx$crossx)，它容易混淆，因为plot的两个参数的顺序与lm公式中的是相反的。

另请参阅

参见第11章更多关于线性回归和lm函数的内容。

10.7 多变量散点图的绘制

问题

数据集包含多个数值型变量。你想得到所有变量对的散点图。

解决方案

将数据放置在一个数据框里，然后绘制该数据框。R为每个列数据对创建一个散点图：

```
> plot(dfrm)
```

讨论

当有大量的变量时，要发现它们之间的相互关系是很困难的。一个有用的技巧是观察所有变量对的散点图。如果逐对绘制，这将是非常繁琐的，R提供了一个简单的方法来一次性产生所有这些散点图。

数据集iris包含4个数值型变量和一个分类变量：

```
> head(iris)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

如果有的话，数值型的列之间存在什么关系呢？绘制这4列生成了多个散点图，它们如图10-7所示。

```
> plot(iris[,1:4])
```

10.8 创建每个因子水平的散点图

问题

数据集包含（至少）两个数值型变量和一个因子。需要为数值型变量创建多个散点图，其中每个散点图对应于一个因子水平。

解决方案

这种图称为条件化图（conditioning plot），它由函数coplot产生：

```
> coplot(y ~ x | f)
```

这将产生*x*对应于*y*的散点图，每幅图对应*f*一个水平。

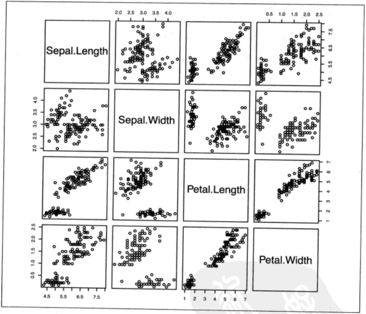


图10-7：多个散点图

讨论

条件化图是另一种探索和说明因子对数据影响的方法。

数据集`Cars93`包含27个变量，它们描述了1993年的93个汽车型号。两个数值型变量是`MPG.city`（它表示在城市中每加仑行驶的里程数）和`Horsepower`（它表示发动机的马力）。一个分类变量是`Origin`，根据汽车在哪里制造，它的值可以是美国制造（USA）或者非美国制造（non-USA）。

探索每加仑英里数（MPG）和马力之间的关系，我们可能会问：美国型号和非美国型号的汽车是否有不同的关系？图10-8中的条件化图显示了MPG对应于马力的两个散点图，它们通过一次性调用函数`coplot`同时创建。左边的图对应美国汽车，右边的图对应非美国汽车。它们由如下代码产生：

```
> data(Cars93, package="MASS")
> coplot(Horsepower ~ MPG.city | Origin, data=Cars93)
```

函数coplot的参数data允许从数据集Cars93中提取变量。

图10-8揭示了一些规律。如果我们真的渴望拥有300马力的汽车，我们将不得不购买在美国制造的汽车；但如果我们想省油，在非美国型号的汽车中，我们会有更多的选择。

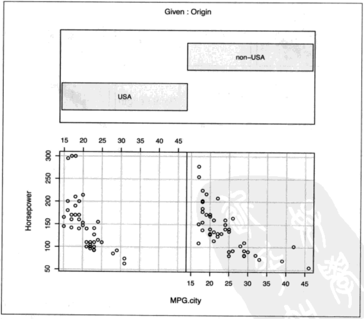


图10-8：条件化图

统计分析可以梳理出这些规律，但可视化表示能更加迅速地揭示它们。

10.9 创建条形图

问题

需要创建一张条形图。

解决方案

调用函数barplot。它的第一个参数是表示条形图高度的向量：

```
> barplot(c(height1, height2, ..., heightn))
```

讨论

函数barplot产生简单的条形图。它假定条形图高度可以方便地存储在一个向量中。然而，事实并非总是如此。经常会有数值型数据向量和为数据分组的平行因子，你想要产生一个表示组平均或组总数的条形图。例如，数据集airquality包含一个数值型Temp列和Month列。可以通过两个步骤根据月份创建平均气温条形图。首先，计算均值：

```
> heights <- tapply(airquality$Temp, airquality$Month, mean)
```

它给出了条形图的高度，通过它来创建条形图：

```
> barplot(heights)
```

结果显示在图10-9a中。正如你所看到的，该结果是很平淡的，所以通常会添加一些简单的装饰：如标题、条块的标签、y轴的标签：

```
> barplot(heights,
+         main="Mean Temp. by Month",
+         names.arg=c("May", "Jun", "Jul", "Aug", "Sep"),
+         ylab="Temp (deg. F)")
```

图10-9b中显示了改善后的条形图。

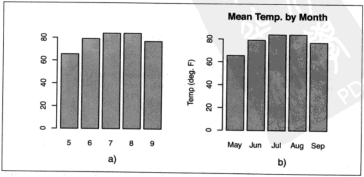


图10-9：无装饰的和有装饰的

另请参阅

关于如何添加置信区间，请参见方法10.10；关于添加颜色的内容，请参见方法10.11；参见软件包lattice中的函数barchart。

10.10 对条形图添加置信区间

问题

需要在一张条形图中增加置信区间。

解决方案

假设x是一个表示统计信息的向量，如一个均值向量，其中lower和upper是该置信区间相应的极限值向量。函数库gplots中的函数barplot2可以显示x的条形图和它的置信区间：

```
> library(gplots)
> barplot2(x, plot.ci=TRUE, ci.l=lower, ci.u=upper)
```

讨论

多数条形图显示点估计，它显示条形的高度。但它们很少包括置信区间。包括我在内的统计学家十分不喜欢这个设置。点估计只能反映一半数据，置信区间才能给出完整的分析。

幸运的是，函数barplot2可以绘制出包含置信区间的条形图。困难的部分是计算区间。在方法10.9中，在绘制它们之前先计算组均值。运用相似的方法，可以先计算出这些均值的置信区间。函数t.test返回一个数值列表。该列表包含一个称为con.int的元素，它本身是一个二元素向量：即均值置信区间的上界值和下界值：

```
> attach(airquality)
> heights <- tapply(Temp, Month, mean)
> lower <- tapply(Temp, Month, function(v) t.test(v)$conf.int[1])
> upper <- tapply(Temp, Month, function(v) t.test(v)$conf.int[2])
```

可以通过这个方法创建一个包含置信区间的基础条形图：

```
> barplot2(heights, plot.ci=TRUE, ci.l=lower, ci.u=upper)
```

但正如在方法10.9中提到的，这会创建一个没有装饰的图形。如果限制y的范围（ylim），控制条形块的宽度（xpd），添加一个标题（main），给条形块添加标签（names.arg），给y轴上标签，条形图将更有吸引力。图10-10中的条形图由调用barplot2产生：

```
> barplot2(heights, plot.ci=TRUE, ci.l=lower, ci.u=upper,
+          ylim=c(50,90), xpd=FALSE,
+          main="Mean Temp. By Month",
+          names.arg=c("May","Jun","Jul","Aug","Sep"),
+          ylab="Temp (deg. F)")
```

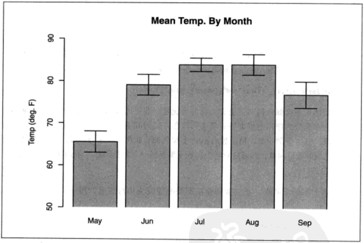


图10-10：包含置信区间的条形图

另请参阅

关于安装gplots软件包的内容，请参见方法3.9；更多关于函数tapply的内容，请参见方法6.5；关于t.test的内容，请参见方法9.9。

10.11 给条形图上色

问题

需要为条形图中的条形块上色或者涂上阴影。

解决方案

使用函数barplot的参数col：

```
> barplot(heights, col=colors)
```

这里，参数heights是条形块高度向量，而参数colors是它相应的颜色向量。

为生成颜色向量，通常会调用函数gray生成灰色向量或调用函数rainbow生成彩色向量。

讨论

构建彩色向量是非常棘手的。简单的解决方案是使用明确的颜色。以下例子绘制了一个有三个条形块的条形图，并且分别将三个条形染为红、白、蓝三色：

```
> barplot(c(3,5,4), col=c("red","white","blue"))
```

然而，生活很少会如此简单。更有可能的情况是，需要通过颜色传达一些关于数据集的信息。一个典型的作用是根据等级将条形涂色：较短的条形块为浅色；较长的条形块的颜色较深。在这个例子中，用函数gray产生灰色调的颜色。函数gray的一个参数是一个在0～1之间的数值向量。该函数对每个向量元素返回一个灰色阴影，它从0.0（纯黑色）～1.0（纯白色）。

为了给条形图涂上阴影，首先要转换条形块的等级为相对高度，用一个0～1之间的值表示：

```
> rel.hts <- rank(heights) / length(heights)
```

然后在颠倒相对高度时，转换相对高度为灰色调向量，这样更高的条形块就是深色而非浅色的：

```
> grays <- gray(1 - rel.hts)
```

可以简单地用这个方法创建涂了阴影的条形图：

```
> barplot(heights, col=grays)
```

然而，对条形图添加装饰使它更易理解。下面是完整的解决方案，结果如图10-11所示。

```
> rel.hts <- (heights - min(heights)) / (max(heights) - min(heights))
> grays <- gray(1 - rel.hts)
> barplot(heights,
+         col=grays,
+         ylim=c(50,90), xpd=FALSE,
+         main="Mean Temp. By Month",
+         names.arg=c("May", "Jun", "Jul", "Aug", "Sep"),
+         ylab="Temp (deg. F)")
```

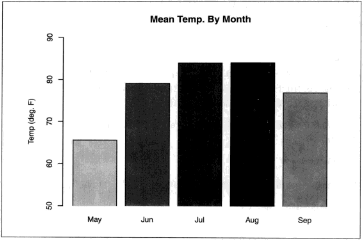


图10-11：填充阴影的条形图

这里用灰色调，因为本书的一些版本没有彩色印刷。为得到一个多色彩的效果，我建议调用函数rainbow，它可以产生一系列颜色。可以调用它来代替这里的gray函数来处理该问题。

另请参阅

关于创建条形图的内容，请参见方法10.9。

10.12 绘制过点*x*和*y*的线

问题

给定成对的观测值：$(x_1, y_1)(x_2, y_2), ..., (x_n, y_n)$。需要绘制一系列线段来连接这些数据点。

解决方案

调用函数plot，它的图形类型为"l"（字母"l"，非数字"1"）。

```
> plot(x, y, type="l")
```

如果数据以一个两列数据框出现，就可以用数据框内容绘制线段：

```
> plot(dfrm, type="l")
```

讨论

这个方法与创建散点图的方法10.1做出了相同的假设——有成对的观测值。绘制散点图与线的唯一区别在于前者绘制点而后者连接它们。函数plot处理这两种情况。参数type决定绘制散点图还是画线，默认情况下是散点图。

可以以两种方式绘制内置的数据集pressure：作为散点图和线图。图10-12a显示散点图，创建命令如下。

```
> plot(pressure)
```

图10-12b是一个线图，它通过简单地改变图形类型为“线”（l）来创建：

```
> plot(pressure, type="l")
```

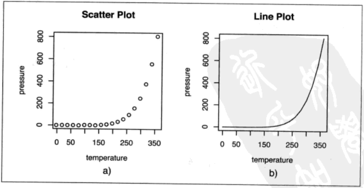


图10-12：相同数据集的两种图形类型

另请参阅

参见方法10.1。

10.13 改变线的类型、宽度或者颜色

问题

你正在绘制一条线，需要改变它的类型、宽度或者颜色。

解决方案

函数plot（以及其他图形函数）含有用于控制线的外观的参数。调用参数lty来控制线的类型：

- lty="solid"或者lty=1（默认）
- lty="dashed"或者lty=2
- lty="dotted" 或者lty=3
- lty="dotdash"或者lty=4
- lty="longdash"或者lty=5
- lty="twodash"或者lty=6
- lty="blank"或者lty=0（抑制绘图）

例如，调用plot以虚线类型画线：

```
> plot(x, y, type="l", lty="dashed")
```

调用参数lwd来控制线的宽度或粗细。根据默认设置，线的宽度为1：

```
> plot(x, y, type="l", lwd=2)              # Draw a thicker line
```

调用参数col来控制线的颜色。根据默认设置，将线画为黑色：

```
> plot(x, y, type="l", col="red")          # Draw a red line
```

讨论

本解决方案演示了如何绘制一条线并指定它的类型、宽度或颜色。常见情况涉及绘制多条线，每条线有它自己的类型、宽度或颜色。该解决方案可以推广到这种情况，它调用函数plot绘制第一条线，然后调用函数lines绘制后续的线。函数lines将线添加到现有的图形中并允许指定它们的类型、宽度和颜色：

```
> plot(x, y.democr, type="l", col="blue")
> lines(x, y.republ, col="red")
> lines(x, y.indeps, col="yellow")
```

这里，最初调用函数plot初始化图形窗口并绘制几条蓝线。后续调用函数lines画出附加的线，第一条为红色，其他为黄色。

另请参阅

更多关于绘制基础线条的内容，请参见方法10.12。

10.14 绘制多个数据集

问题

需要在一个图形中显示多个数据集。

解决方案

使用高级图形函数，如plot或者curve来初始化图形。然后使用低级图形函数，如lines和points来添加额外的数据集。

当初始化图形时，小心要正确地设定画布的大小。使用参数xlim和ylim，创建大到足以容纳所有数据集而非仅仅是第一个数据集的画布。下面的代码片段显示了如何使用函数range计算极限值，然后使用xlim和ylim来设置它们。假定有两个数据集，一个由x1和y1代表，另一个由x2和y2代表：

```
> xlim <- range(c(x1,x2))
> ylim <- range(c(y1,y2))
> plot(x1, y1, type="l", xlim=xlim, ylim=ylim)
> lines(x2, y2, lty="dashed")
```

讨论

正如在本章“简介”所讨论的，函数plot初始化图形窗口，然后函数lines添加线。我第一次尝试同时使用图形和线的情况是这样的（我想同时绘制(x1, y1)数据集和(x2, y2)数据集）：

```
> plot(x1, y1, type="l")
> lines(x2, y2, lty="dashed")
```

该结果类似于图10-13a。第一个数据集绘制正确，但第二个数据集有些落入页面外面。这种表示会令人迷惑。

问题在于图形函数根据给定的数据来设定画布大小。第二个数据集（由函数lines绘制）有不同的、更广的范围，所以它的图形一部分落入画布外，看起来消失了。

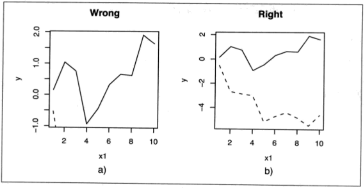


图10-13：正确和错误的多数据集绘图

解决方案是首先正确地设定画布大小。函数plot有两个用来设定画布大小的参数——xlim是一个有两个元素的向量，它给出了x轴的下限和上限，而ylim给出了y轴的下限和上限：

```
> plot(x, y, xlim=c(lower,upper), ylim=c(lower,upper), ...)
```

这里调用函数range将十分便利。它返回一个向量的上限和下限，因此可以在调用plot前，用它来计算xlim和ylim，如下所示。

```
> xlim <- range(c(x1,x2))
> ylim <- range(c(y1,y2))
> plot(x1, y1, type="l", xlim=xlim, ylim=ylim)
> lines(x2, y2, lty="dashed")
```

图10-13b显示了该结果。现在画布足够大了，没有数据丢失。

“讨论”中的内容是针对线来讨论的，但绘制点也具有相同的问题，因此有相同的解决方案。

另请参阅

参见本章“简介”中关于高级图形函数和低级图形函数的评论。

10.15 添加垂直线和水平线

问题

需要对图形添加垂直线或者水平线，例如过原点的一条轴。

解决方案

当给定一个参数v或h时，函数abline会分别画出一条垂直线或者水平线：

```
> abline(v=x)          # Draw a vertical line at x
> abline(h=y)          # Draw a horizontal line at y
```

讨论

函数abline画出的是直线。我通常调用它来画经过图形原点的轴：

```
> abline(v=0)          # Vertical line at x = 0
> abline(h=0)          # Horizontal line at y = 0
```

函数abline将线添加到现有的图形中，它不能创建一个新的图形。所以首先调用一个图形创建函数，如plot(...)，然后再调用abline。

参数v和h可以是向量，在这种情况下，abline绘制多条线——每条线对应一个向量元素。一个典型的用法是有规律地画出间隔线。假设有一个点样本samp。首先，绘制这些点，然后绘制一条经过它们均值的实线：

```
> plot(samp)
> m <- mean(samp)
> abline(h=m)
```

我们想要用图形来说明样本标准差，所以我们计算它并在距离均值±1和±2个标准差的地方绘制虚线：

```
> stdevs <- m + c(-2,-1,+1,+2)*sd(samp)
> abline(h=stdevs, lty="dotted")
```

结果如图10-14所示。

另请参阅

更多关于改变线类型的内容，请参见方法10.13。

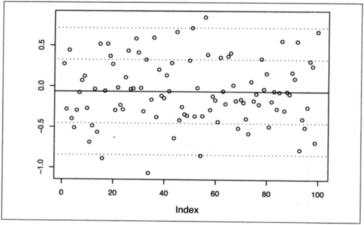


图10-14：描绘样本标准差

10.16 创建箱线图

问题

绘制数据的箱线图。

解决方案

调用boxplot(x)，其中x是一个数值向量。

讨论

箱线图对一个数据集提供了的快速简便的可视化总结。图10-15显示了一个典型的箱线图：

- 中间的粗线是中位数。
- 中位数周围的方框标识了第一个和第三个四分位数，方框底端是Q1，顶端是Q3。
- 方框上方和下方的“虚线”显示了数据的范围，不包括离群值（outlier）。
- 圆圈识别了离群值。在默认情况下，离群值定义为任何超出方框之外1.5 × IQR的数值。（IQR为四分位差，或Q3-Q1）。在这个例子中，有三个离群值。

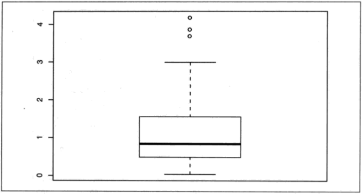


图10-15：典型的箱线图

另请参阅

单一的箱线图是很乏味的。创建多个箱线图的方法请参见方法10.17。

10.17 对每个因子水平创建箱线图

问题

数据集包含一个数值型变量和一个因子（分类变量）。你想创建多个根据因子水平分组的数值型变量的箱线图。

解决方案

调用含有公式的函数boxplot：

```
> boxplot(x ~ f)
```

这里，x是数值型变量，f是因子。

也可以调用函数plot的两参数形式，注意第一个参数是因子：

```
> plot(f, x)
```

讨论

这个方法是另一个探索和说明两变量间关系的好方法。在这种情况下，我们想知道数值型变量是否根据因子水平而变化。

数据集UScereal包含许多有关早餐麦片的变量。其中一个变量代表每份的含糖量，另一个代表货架位置（从地面向上数）。谷物制造商可以为了货架位置谈判，把他们的产品放在最有销售潜力的位置。我寻思：他们把含糖量高的谷类食品放在哪里呢？可以通过对每个货架创建一个箱线图来探索这个问题，如下所示。

```
> data(UScereal, package="MASS")
> boxplot(sugars ~ shelf, data=UScereal)
```

然而，这个箱线图十分单调，所以添加一些修饰：

```
> data(UScereal, package="MASS")
> boxplot(sugars ~ shelf, data=UScereal,
+         main="Sugar Content by Shelf",
+         xlab="Shelf", ylab="Sugar (grams per portion)")
```

结果如图10-16所示。参数data让我们从数据框UScereal中提取变量。因为基本箱线图没有标签，使它很难解释，所以我给出了xlab（x轴标签）和ylab（y轴标签）。

该箱线图表明货架#2有含糖量最高的谷物。难道这是由于这个货架在儿童视觉高度的位置而孩子会影响他们父母的谷物消费选择？

另请参阅

关于创建一个基本箱线图的内容，请参见方法10.16。

10.18 创建直方图

问题

需要创建数据的直方图。

解决方案

调用函数hist(x)，其中x是一个数值向量。

讨论

图10-17a显示了数据集Cars93中的MPG.city列的直方图，它的创建代码如下：

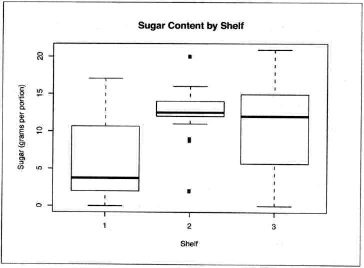


图10-16：根据因子水平的箱线图

```
> data(Cars93, package="MASS")
> hist(Cars93$MPG.city)
```

在函数hist中，必须决定为了容纳数据需要创建多少单元（直方块）。在这个例子中，默认的算法选择了7个直方块。因为分布的形状还是未知的，所以对于我的情况它创建了过少的条形块。所以在hist中包含了第二个参数——即建议的直方块数量：

```
> hist(Cars93$MPG.city, 20)
```

这个数目只是一个建议，但hist会尽可能地扩张直方块的数量以适应这个建议。

图10-17的右边的面板显示了相同数据的另一个直方图，它有更多的直方块，而且它的默认标题以及x轴标签修改了。它的创建代码如下：

```
> hist(Cars93$MPG.city, 20, main="City MPG (1993)", xlab="MPG")
```

另请参阅

软件包lattice的函数histogram是hist的另一种表达方式。

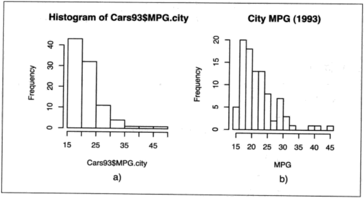


图10-17：直方图

10.19 对直方图添加密度估计

问题

现在已经有了数据的直方图，而需要添加一条曲线以显示数据所体现的密度。

解决方案

调用函数density来近似估计样本密度，然后调用lines画出这个近似密度：

```
> hist(x, prob=T)              # Histogram of x, using a probability scale
> lines(density(x))            # Graph the approximate density
```

讨论

直方图显示数据的密度函数，但它的计算是粗糙的。一个更顺畅的估计可以帮助你更好地可视化它所服从的分布。

这个例子从伽玛分布中抽取样本，然后绘制直方图和估计的密度。结果如图10-18所示。

```
> samp <- rgamma(500, 2, 2)
> hist(samp, 20, prob=T)
> lines(density(samp))
```

为了使这个方法奏效，必须在hist中包括参数prob=T。参数prob=T的结果是图10-18的y轴由概率形式给出。没有它，函数hist给出的是计数，这将与绘制density的输出不相容。

另请参阅

密度函数用非参数方法近似估计密度的形状。如果你知道真实的基础分布，那么使用方法8.11来绘制密度函数。

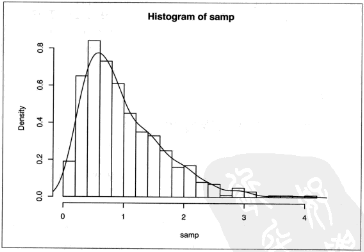


图10-18：有密度估计的直方图

10.20 创建离散直方图

问题

需要创建离散数据的直方图。

解决方案

调用函数table来对事件进行计数。然后调用包含参数type="h"的plot函数来将事件绘制成一个直方图：

```
> plot(table(x), type="h")
```

这里，x是一个离散值向量。

讨论

可以调用函数hist来创建一个关于离散数据的直方图，但是函数hist其实是面向连续数据的。结果中的直方图没有“离散”的外表。

图10-19显示了调用此函数而创建的直方图：

```
> plot(table(x), type="h", lwd=5, ylab="Freq")
```

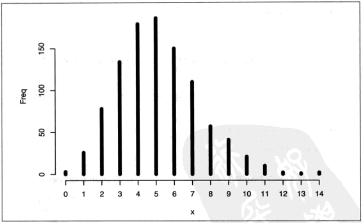


图10-19：离散数据的直方图

参数lwd=5稍微扩大了条形块的宽度，给它们更赏心悦目的外观。

如果你喜欢有相对频率的直方图，且不是计数，那么你可以简单地通过数据项的数目来缩放表格元素，如下所示。

```
> plot(table(x)/length(x), type="h", lwd=5, ylab="Freq")
```

另请参阅

关于连续数据的直方图，请参见方法10.18。本方法直接取自plot的帮助页。

10.21 创建正态Q-Q图

问题

需要创建数据的Q-Q图。通常用Q-Q图来判断数据是否服从正态分布。

解决方案

调用函数qqnorm来创建基本Q-Q图，然后调用qqline对它添加一条对角线：

```
> qqnorm(x)
> qqline(x)
```

这里，x是一个数值向量。

讨论

有时候，知道数据是否为正态分布是很重要的。绘制Q-Q图是初步检验正态性的一个很好的方法。

数据集Cars93包含一个Price（价格）列。它服从正态分布的吗？以下代码片段创建Price数据的Q-Q图，它如图10-20a所示。

```
> data(Cars93, package="MASS")
> qqnorm(Cars93$Price, main="Q-Q Plot: Price")
> qqline(Cars93$Price)
```

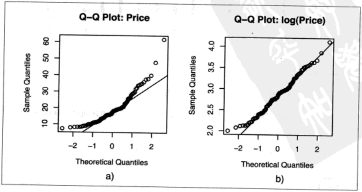


图10-20：Q-Q图

如果数据服从一个完美的正态分布，那么点会精确地落在对角线上。在图10-20a中，许多点很接近对角线，尤其是在中间部分的点，尾部的点就离对角线很远了，太多的点在对角线之上，表示大体向左偏的趋势。

向左偏斜也许能够通过对数变换纠正。可以绘制log(Price)，它将产生图10-20b所示的图形。

```
> data(Cars93, package="MASS")
> qqnorm(log(Cars93$Price), main="Q-Q Plot: log(Price)")
> qqline(log(Cars93$Price))
```

注意，在新图形中的点有更理想的态势，除了左端尾部的极端位置以外，大部分点与对角线保持很近距离。这表明log(Price)是近似正态的。

另请参阅

关于创建其他分布Q-Q图的内容，请参见方法10.22。关于应用正态Q-Q图识别线性回归的内容，请参见方法11.15。

10.22 创建其他Q-Q图

问题

查看非正态分布数据的Q-Q图。

解决方案

在这个方法中，需要先具备有关分布的一些知识。解决方案由以下步骤构成：

- 调用函数ppoints生成一个在0～1之间的点序列。
- 将这些点转化成分位数，使用假定分布的分位数函数。
- 将样本数据进行排序。
- 绘制计算的分位数和排序后样本数据的散点图。
- 调用函数abline绘制对角线。

以上步骤都可以在两行R代码中完成。下面是一个例子，它假设数据y有自由度为5的学生*t*分布。前面讲过学生*t*分布的分位数函数是qt，而它的第二个参数是自由度：

```
> plot(qt(ppoints(y), 5), sort(y))
> abline(a=0, b=1)
```

讨论

这个解决方案看似复杂，其实不然。R代码通常如下所示：

```
> plot(quantileFunction(ppoints(y), ...), sort(y))
> abline(a=0, b=1)
```

其中，*quantileFunction*是假定分布的分位数函数，而"..."省略的是*quantileFunction*所需要的y以外的参数。函数abline沿着对角线绘制一条线。在理想情况下，所有数据点正好落在该直线上。

以下举例来说明这个方法。将从均值为10（或比率为1/10）的指数分布中随机抽样：

```
> RATE <- 1/10
> y <- rexp(N, rate=RATE)
```

对于指数分布的分位数函数是qexp，它有一个参数rate。可以用以下方法创建Q-Q图：

```
> plot(qexp(ppoints(y), rate=RATE), sort(y))
> abline(a=0, b=1)
```

这时得到的图形看起来会过于平淡，所以包含一个标题和标签来帮助读者理解它。以下是创建图10-21的实际代码：

```
> plot(qexp(ppoints(y), rate=RATE), sort(y),
+       main="Q-Q Plot", xlab="Theoretical Quantiles", ylab="Sample Quantiles")
> abline(a=0, b=1)
```

观察Q-Q图上的点可以偏离对角线多远是很件有趣的事——即使我们已经知道准确的总体分布。

10.23 用多种颜色绘制变量

问题

需要用多种颜色绘制数据，这往往会使图形包含更多信息，更易读或者更有趣。

解决方案

在调用函数plot时，应用参数col：

```
> plot(x, col=colors)
```

参数col的值可以是：

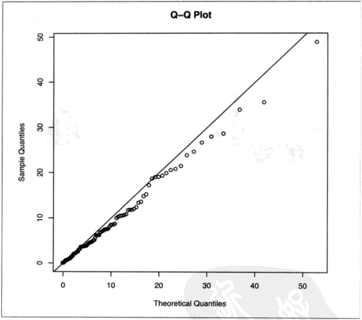


图10-21：指数分布下的Q-Q图

- 一种颜色，在这种情况下，所有数据点都是这种颜色。
- 一个颜色向量，与x有相同的长度，在这种情况下，x的每个值有它相应的颜色。
- 一个短向量，在这种情况下，颜色向量是可循环的。

讨论

如果使用如下代码绘制x，那么它的图形是黑色的：

```
> plot(x)
```

可以用蓝色绘制整个图形，如下所示。

```
> plot(x, col="blue")
```

然而，让颜色为说明数据而改变会更加有用和有趣。通过使用两种方式绘制一个图形来说明这点，一次用黑白两色绘图，而另一次只用简单的阴影。调用plot会产生基本的黑白图，它如图10-22a所示。

```
> plot(x, type="h", lwd=3)
```

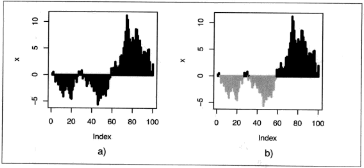


图10-22：对图形添加阴影

（注意，这里设置函数plot的参数lwd=3，它拓宽了线条，使它们更容易注意到。）

观察图10-22b，它先根据x的符号生成一个colors向量，向量的元素取值为"gray"或"black"，然后使用向量colors的颜色绘制x：

```
> colors <- ifelse(x >= 0, "black", "gray")
> plot(x, type='h', lwd=3, col=colors)
```

现在，把负值绘制成灰色，这是因为colors中相应的元素是"gray"。

使用灰色调是因为本书的一些版本是黑白印刷的，所以读者看不出彩色。可以很容易地使用真正的全色彩来代替——例如，通过这个赋值语句改变colors的定义：

```
> colors <- ifelse(x >= 0, "green", "red")
```

这将绘制正值为绿色，负值为红色。

奇怪的是，当画线时，这个方法不奏效了。调用plot可能看似是在绘制一条直线，它的颜色在蓝色和绿色之间交替：

```
> plot(x, type="l", col=c("blue", "green"))
```

但结果不尽如人意。第一种颜色"blue"用于所有线段，而第二种颜色"green"被忽略了。

另请参阅

参见方法5.3关于循环规则的内容。执行colors()得出一列可用颜色，并调用函数segments用多种颜色绘制线段。

10.24 绘制函数

问题

需要绘制一个函数的值。

解决方案

在给定函数和它的定义域范围的情况下，函数curve可以绘制函数：

```
> curve(sin, -3, +3)            # Graph the sine function from -3 to +3
```

讨论

图10-23a显示了标准正态密度函数的图。该图创建如下。

```
> curve(dnorm, -3.5, +3.5,
+       main="Std. Normal Density")
```

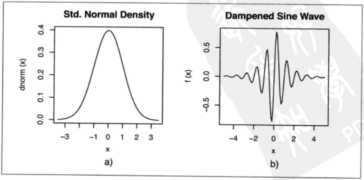


图10-23：绘制函数

函数curve调用函数dnorm，并设定参数范围在-3.5～+3.5之间，然后绘制这个结果。正如任何功能强大的高级图形函数一样，它接受一个参数main来指定标题。

curve可以绘制任何接受一个参数并且返回一个值的函数。图10-23b是通过定义一个区域函数并绘制它来创建：

```
> f <- function(x) exp(-abs(x)) * sin(2*pi*x)
> curve(f, -5, +5, main="Dampened Sine Wave")
```

另请参阅

关于如何定义一个函数的内容，参见方法2.12。

10.25 图形间暂停

问题

在创建多个图形时，每个新创建的图形都覆盖了先前的图形。需要R在两图形间暂停，这样就可以在每个图形被覆盖前查看它们。

解决方案

有一个称为ask的全局图形选项。将它设置为TRUE，R会在每个新图形前暂停。

```
> par(ask=TRUE)
```

当你对R在图形间暂停感到厌烦时，可以设置它为FALSE：

```
> par(ask=FALSE)
```

讨论

当ask设定为TRUE时，R在开始一个新图形前会马上输出以下信息：

```
Waiting to confirm page change...
```

当你准备好绘制新图形时，单击图形窗口。R会开始下一个图形。

另请参阅

如果一个图形覆盖了另一个，考虑使用方法10.26中关于在一个框架中绘制多个图形的内容。更多关于改变图形参数的内容，请参见方法10.29。

10.26 在一页中显示多个图形

问题

需要在一页中并排显示多个图形。

解决方案

首先，把图形窗口分为*N*行和*M*列的矩阵。这个步骤可以通过设置称为`mfrow`的图形参数完成。它的值是一个二元素向量，它给出了行和列的数目：

```
> par(mfrow=(c(N,M))          # Divide the graphics window into N x M matrix
```

其次，通过反复调用`plot`来填充矩阵中的每个单元。例如，在创建一个3 × 2网格后，可以调用6次`plot`函数来填满整个网格。

讨论

让我们用一张包含多个图形的画布来显示4种不同的beta分布。首先，在一个2 × 2网格中创建4个绘图区：

```
> par(mfrow=c(2,2))
```

其次，调用四次`plot`函数，每次绘制一个不同的beta分布[注1]

```
> Quantile <- seq(from=0, to=1, length.out=30)
> plot(Quantile, dbeta(Quantile, 2, 4), type="l", main="First")
> plot(Quantile, dbeta(Quantile, 4, 2), type="l", main="Second")
> plot(Quantile, dbeta(Quantile, 1, 1), type="l", main="Third")
> plot(Quantile, dbeta(Quantile, 0.5, 0.5), type="l", main="Fourth")
```

结果如图10-24所示。每个图形都有它们自己的标题（例如，`main="First"`），这是我用来说明绘图顺序的。每个图形还可以有它们各自的标签、说明、网格和其他装饰。

当调用`mfrow`创建该图形网格矩阵时，R逐行填充图形窗口。在刚才所举的例子中，R先填充第一行，然后是第二行。你可能更喜欢逐列填充它。此时可以采用一种称为`mfcol`的图形参数，就可以逐列填充了。它的工作原理与`mfrow`相似，但促使R逐列填充矩阵。调用`par`把图形窗口分为3行3列的矩阵：

```
> par(mfcol=c(3,3))           # mfcol, not mfrow
```

注1：　函数dbeta用来计算beta分布的概率密度函数。第二和第三个参数是分布的形状参数。

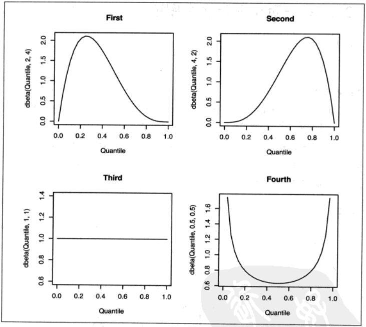


图10-24：几个beta分布的图形

连续地调用plot将填充第一列，然后第二列，并最终填充第三列。

参数mfrow和mfcol让你创建一个简单的图形矩阵。对于更复杂的设计，考虑调用函数layout和函数split.screen。

另请参阅

方法8.11展示了一个绘制多个图形的类似例子。另一个显示多个图形的方法是通过打开多个图形窗口，具体参见方法10.27。更多关于函数par和图形参数的内容，请参见方法10.29。

软件包grid和lattice包含额外的显示多图形的工具。

10.27 打开另一个图形窗口

问题

你需要打开另一个图形窗口，这样你就能同时查看两个图形。

解决方案

调用函数win.graph以打开第二个图形窗口：

```
> win.graph()
```

随后的图形输出将出现在新的窗口中。原来的图形窗口将保持不变。

讨论

每次创建一个新的图形，它都会覆盖先前图形窗口中的内容。有时你想在保留现有图形的同时，创建一个新的图形，通常这样你就可以并排比较它们。

函数win.graph创建一个新的图形窗口。新窗口成为当前活动窗口，这意味着所有的图形输出由它管理。其他所有的图形窗口变得不活跃，它们的内容被冻结。

最终，你可能要冻结活动窗口，切换到一个的非活动窗口并覆盖它。函数dev.set在窗口间切换。活动窗口变为非活动的，而下一个窗口（按数字顺序）成为活动窗口：

```
> dev.set()
windows
      2
```

函数dev.set返回现有活动窗口的数目。

在默认情况下，R建立一个方形窗口。你可以调用win.graph设置它的宽度和高度参数，从而改变默认的尺寸。我更喜欢图形尺寸基于黄金比例（1.618），例如，我经常创建新的图形窗口如下[注2]：

```
> win.graph(width=7.0, height=5.5)
```

最后，注意，win.graph是一个独立于系统的便捷函数：它适用于所有平台，但它仅仅是一个依赖于系统函数（用于实际创建新图形窗口）的包装。如果你使用依赖于系统的

注2： 的确，我知道7.0/5.5不是黄金比例。参数width和height是外框的尺寸。由于边框的调整，（内部）区域面积会小一些。这个宽度和高度在我的计算机上产生一个有着适当尺寸的图形区域。这个方法对你可能会不奏效。

函数，你将有更多的控制，但你的代码就不是便捷的。参见帮助说明页有关win.graph的内容。

另请参阅

参见方法10.26关于在一个窗口中创建多个图形的内容。

10.28 在文档中绘制图形

问题

你需要把图形保存在一个文件中，例如PNG、JPEG或PostScript文件。

解决方案

在屏幕上创建图形，然后调用函数savePlot：

```
> savePlot(filename="filename.ext", type="type")
```

参数*type*是平台专用的。参见Devices帮助说明页，查看你机器中可用的类型列表：

```
> help(Devices)
```

大部分平台允许"png"类型或"jpeg"类型的***type***，但每个平台也有其他的类型。

讨论

在Windows操作系统中，快捷的方法是选定图形窗口，然后点击菜单中的“文件”（File）→“另存为”（Save as...）。它会在写入文件前，提示你输入文件类型和文件名。

这个过程与OS X操作系统相似。

记住：该文件将写入当前的工作目录（除非你使用绝对文档路径），所以你在调用savePlot之前，先确定是哪个目录（参见方法3.1）。

该解决方案在批量处理脚本中会有所不同：

- 调用一个函数来打开一个新的图形文件，例如png(...)或jpeg(...)。
- 调用函数plot和它的参数来生成图形。
- 调用函数dev.off()关闭图形文件。

打开图形文件的函数取决于你的平台和文件格式。再次参见help(Devices)给出的文档，查看你平台上的可用函数列表。一个常见的函数是png，它创建一个PNG文件。它的使用方法如下：

```
> png("filename.ext", width=w, height=h)
```

这里，*w*是期望的宽度，而*h*是期望的高度，它们都由像素表示。调用这个函数，该方案会如下执行并创建图形文件*myPlot.png*：

```
> png("myPlot.png", width=648, height=432)        # Or whatever dimensions work for you
> plot(x, y, main="Scatterplot of X, Y")
> dev.off()
```

另请参阅

更多关于当前工作目录的内容，请参见方法3.1。

10.29 改变图形参数

问题

你需要改变图形软件的一个全局参数，例如线型、背景颜色，或者字体大小。

解决方案

调用函数par，它让你设置全局图形参数的值。例如，以下par的调用会改变默认线宽从1到2：

```
> par(lwd=2)
```

讨论

每个图形取决于很多假设。线应该有多宽？背景颜色是什么？前景颜色是什么？字体是多大？边距必须有多宽？假设答案是由全局图形参数和它们的默认值给出的。多亏那些默认值；否则，最简单的图像也将因为设置每个细节而变得麻烦。

然而，有时默认设置并不适合你，你需要改变它们。函数par的作用就在于此。R中有几十个这样的参数，而严谨的图形爱好者会研究在par的帮助说明页中出现的完整列表。

为了查看一个参数的当前值，调用par和它的字符串参数名：

```
> par("lty")              # What is the default line type?
[1] "solid"
```

```
> par("bg")              # What is the default background color?
[1] "transparent"
```

为了设置一个参数值，把它作为一个函数参数传递给函数par。表10-1列出了一些常见用法。

表10-1：函数par的参数

参数	作用	例子
ask=*logical*	如果为TRUE，在每个新图前暂停（方法10.25）	par(ask=TRUE)
bg="*color*"	背景色	par(bg="lightyellow")
cex=*number*	文字或者绘图点的高度，表示为标准尺寸的倍数	par(cex=1.5)
col="*color*"	默认绘图颜色	par(col="blue")
fg="*color*"	前景色	par(fg="gray")
lty="*linetype*"	线型：实线、虚线等	par(lty="dotted")
lwd=*number*	线宽：1：正常；2：宽；3：更宽	par(lwd=2)
mfcol=c(*nr*,*nc*) 或者 mfrow=c(*nr*,*nc*)	建立多图画布矩阵，*nr*行，*nc*列（方法10.26）	par(mfrow=c(2,2))
new=*logical*	在一张图上绘制另外一张图形	par(new=TRUE)
pch=*pointtype*	默认点类型（参见函数points的帮助页面）	par(pch=21)
xlog=*logical*	采用*x*轴的对数标度	par(xlog=TRUE)
ylog=*logical*	采用y轴的对数标度	par(ylog=TRUE)

然而，函数par伴随着一个警告。

警告： 改变一个全局参数自然有一个全局影响。这个变化会影响所有的图形，不只是下一个，除非你把这个改变再改回原来的默认值。此外，如果你调用R软件包来创建它们自己的图形，那么这个全局变化也将影响它们。所以，在做出这些全局改变前，三思而后行：真的想在所有的图形中对所有的线都设置成（例如）：洋红色的、虚线的、三倍宽？可能不会，所以尽可能使用局部参数而不是全局参数。

另请参阅

函数par的帮助说明页列出了全局图形参数；《R in a Nutshell》中关于图形的章节包含了有用注释的列表。《R Graphics》包含了图形参数的大量解释。

第11章

线性回归和方差分析

简介

在统计学中，建模是我们主要研究的内容。模型量化了变量之间的关系，也可以给我们做出预测。

简单线性回归是最基本的模型。它只有两个变量，它用一个含有误差项的线性关系来建模：

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

给定数据x和y，我们的任务是拟合模型，即给出β_0和β_1的最佳估计（参见方法11.1）。

可以自然地把简单线性回归推广到多元线性回归。多元线性回归有多个变量在关系式的右侧（参见方法11.2），即

$$y_i = \beta_0 + \beta_1 u_i + \beta_2 v_i + \beta_3 w_i + \varepsilon_i$$

统计学家把u、v和w称为预测变量，而y称为响应变量（即因变量）。显然，只有在预测变量和响应变量之间有相应的线性关系时，该模型才是有用的。但这一要求比你想象的要灵活许多。方法11.11讨论把变量转化为一个（或者更多）线性关系，这样一来，你就可以使用广泛应用的线性回归模型了。

R的美妙之处在于任何人都可以建立这些线性模型。模型由函数`lm`建立，它返回一个模型对象。从模型对象中，我们得到系数（β_i）和回归统计量。这个操作很容易。

R的不便之处也在于任何人都可以建立这些模型。没有要求你检查该模型是否合理，统

计是否显著。在你盲目相信模型前，应该先检验它。你需要的大多数信息在回归结果的总结中（参见方法11.4）：

模型统计显著吗

检验回归结果总结下面的*F*统计量部分。

回归系数是否显著

检验在回归结果总结中系数的*t*统计量和*p*值，或检验它们的置信区间（参见方法11.13）。

模型是否有用

检验接近总结底部的R^2。

模型是否很好地拟合数据

绘制残差图并检验回归诊断（参见方法11.14和方法11.15）。

数据是否满足应用线性回归应该满足的假设

检验模型诊断，确认你的线性模型是否合理地匹配数据（参见方法11.15）。

方差分析

方差分析（Analysis of Variance，ANOVA）是一个功能强大的统计方法。由于它的重要性，无论是理论还是实践，几乎所有统计学一年级的研究生都要学习方差分析。然而，我常常感到惊讶，在多大程度上，非统计学专业的人不知道它的目的和价值。

回归创建了一个模型，而方差分析是评估这些模型的方法之一。方差分析的数学与回归的数学交织在一起，所以统计学家通常将它们一起呈现出来；我在这里遵循这个惯例。

方差分析实际上是与普通数学分析相联系的众多技术之一。本章讲述方差分析的以下几个方面的应用。

单因素方差分析

这是方差分析最简单的应用。假设你有来自多个总体的样本数据，并且想知道这些总体是否有不同的均值。单因素方差分析可以解决这个问题。如果这些总体服从正态分布，则可以调用函数`oneway.test`（参见方法11.20）；否则，调用它的非参数版本，即函数`kruskal.test`（参见方法11.23）。

模型比较

当你从一个线性回归中添加或删除一个预测变量时，你要知道这个改变是否改善了模型。函数`anova`比较两个回归模型，并显示它们是否有显著不同（参见方法11.24）。

方差分析表

函数anova也可以建立线性回归模型的方差分析表，其中包括F统计量，它用于衡量模型的统计显著性（参见方法11.3）。几乎每本关于回归的教科书都讨论了这个重要的表。

以下参考文献包含更多关于方差分析的数学方法。

另请参阅

有许多关于线性回归的书籍。我的最爱之一是《Applied Linear Regression Models (4th ed.)》，它由Kutner、Nachtsheim和Neter撰写（McGraw-Hill/Irwin）。我在本章中遵照他们的术语和惯例。

我也很喜欢Julian Faraway的著作《Practical Regression and ANOVA Using R》，因为它使用R来说明回归并且它具有相当的可读性。你可以从CRAN上免费下载PDF（*http://cran.r-project.org/doc/contrib/Faraway-PRA.pdf*）版。

11.1 简单线性回归

问题

有两个向量x和y，它们包含成对观察值：$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$。你相信$x$和$y$之间有一个线性关系，你想要创建描述这个关系的一个回归模型。

解决方案

函数lm运行一个线性回归并显示它的系数：

```
> lm(y ~ x)
Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
      17.72         3.25
```

讨论

简单线性回归包含两个变量：一个预测变量，通常称为x；和一个响应变量，通常称为y。回归使用普通最小二乘法（Ordinary Least-Squares，OLS）来拟和线性模型：

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$$

其中β_0和β_1是回归系数而ε_i是误差项。

函数lm可以运行线性回归。它的主要参数是一个模型公式，如y ~ x。该公式包含一个波形符（~），左侧为响应变量，右侧为预测变量。函数lm估计回归系数β_0和β_1，并把它们分别作为截距和x的系数显示出来：

```
Coefficients:
(Intercept)           x
      17.72        3.25
```

在这种情况下，回归方程是：

$$y_i = 17.72 + 3.25x_i + \varepsilon_i$$

数据由一个数据框展示是很普遍的，在这种情况下，你想运行两个数据框的列之间的回归关系。这里，x列和y列是数据框dfrm的列：

```
> dfrm
            x         y
1  0.04781401  5.406651
2  1.90857986 19.941568
3  2.79987246 23.922613
4  4.46755305 32.432904
5  3.76490363 44.259268
6  5.92364632 61.151480
7  8.04611587 26.305505
8  7.11097986 43.606087
9  9.73645966 58.262112
10 9.19324543 57.631029
.
. (etc.)
.
```

在函数lm中，参数data用来指定用于线性回归的数据框。这时，函数lm会从数据框而非你的R工作空间来访问用于回归的变量：

```
> lm(y ~ x, data=dfrm)          # Take x and y from dfrm

Call:
lm(formula = y ~ x, data = dfrm)

Coefficients:
(Intercept)           x
      17.72        3.25
```

11.2 多元线性回归

问题

有几个预测变量（例如，u、v和w）以及响应变量y。你相信预测变量和响应变量之间有线性关系，现在需要对数据建立一个线性回归模型。

解决方案

调用函数lm。指定多个公式右侧的预测变量，由加号（+）分开：

```
> lm(y ~ u + v + w)
```

讨论

多元线性回归是简单线性回归的推广。它允许多个预测变量，而不是一个预测变量，而且仍然使用普通最小二乘法来计算线性方程的系数。以下线性模型就是有三个变量的回归：

$$y_i = \beta_0 + \beta_1 u_i + \beta_2 v_i + \beta_3 w_i + \varepsilon_i$$

R中的函数lm既可用于简单线性回归，又可用于多元线性回归。你可以方便地添加多个变量至模型公式的右边。函数的输出显示拟合模型的系数：

```
> lm(y ~ u + v + w)
Call:
lm(formula = y ~ u + v + w)

Coefficients:
(Intercept)            u            v            w
     1.4222       1.0359       0.9217       0.7261
```

当变量数量增加时，函数lm的参数data是特别有用的，它更易于把数据保存在一个数据框中而不使它们成为许多分散的变量。假设你的数据显示在一个数据框中，例如如下显示的变量dfrm：

```
> dfrm
         y           u         v        w
1 6.584519  0.79939065 2.7971413 4.366557
2 6.425215 -2.31338537 2.7836201 4.515084
3 7.830578  1.71736899 2.7570401 3.865557
4 2.757777  1.27652888 0.4191765 2.547935
5 5.794566  0.39643488 2.3785468 3.265971
```

```
6  7.314611  1.82247760 1.8291302 4.518522
7  2.533638 -1.34186107 2.3472593 2.570884
8  8.696910  0.75946803 3.4028180 4.442560
9  6.304464  0.92000133 2.0654513 2.835248
10 8.095094  1.02341093 2.6729252 3.868573
.
. (etc.)
.
```

当我们把数据框dfrm提供给lm的参数data时，R在数据框的列中寻找回归变量：

```
> lm(y ~ u + v + w, data=dfrm)

Call:
lm(formula = y ~ u + v + w, data = dfrm)

Coefficients:
(Intercept)            u         v         w
     1.4222       1.0359    0.9217    0.7261
```

另请参阅

关于简单线性回归的内容，请参见方法11.1。

11.3 得到回归统计量

问题

想得到回归关系的关键统计量和回归方程的相关信息，例如R^2、F统计量、回归系数的置信区间、残差、方差分析表等。

解决方案

把回归模型保存在一个变量中，例如m：

```
> m <- lm(y ~ u + v + w)
```

然后调用函数来从模型中提取回归统计量和相关信息。各个函数的功能如下：

anova(m)
: 给出方差分析表。

coefficients(m)
: 给出模型系数。

`coef(m)`

与`coefficients(m)`相同。

`confint(m)`

给出回归系数的置信区间。

`deviance(m)`

给出残差平方和。

`effects(m)`

给出正交影响向量。

`fitted(m)`

给出拟合*y*值的向量。

`residuals(m)`

给出模型残差。

`resid(m)`

与`residuals(m)`相同。

`summary(m)`

给出重要统计量，例如R^2、F统计量和残差标准误差(σ)。

`vcov(m)`

主要参数的方差——协方差矩阵。

讨论

当我开始使用R时，帮助文档指出使用函数`lm`来运行线性回归。所以我进行了如下的操作并得到如方法11.2中所示的输出：

```
> lm(y ~ u + v + w)
Call:
lm(formula = y ~ u + v + w)
Coefficients:
(Intercept)           u         v         w
     1.4222      1.0359    0.9217    0.7261
```

我很失望！输出结果与其他统计软件包，如SAS，相比缺少很多信息。R^2在哪里？系数的置信区间在哪里？F统计量、它的p值以及方差分析表又在哪里呢？

当然，所有这些信息是可以得到的——你只是必须请求它们。其他统计系统显示一切数

据并让你费力地查阅它们。 R更加注重简约。它打印一个最基本的输出，让你向它要求更多你想知道的信息。

函数lm返回一个模型对象，你可以把它分配给一个变量：

```
> m <- lm(y ~ u + v + w)
```

从模型对象中，你可以通过调用专门的函数来提取重要的信息。最重要的函数是summary：

```
> summary(m)

Call:
lm(formula = y ~ u + v + w)

Residuals:
    Min      1Q  Median      3Q     Max
-3.3965 -0.9472 -0.4708 1.3730 3.1283
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 1.4222 1.4036 1.013 0.32029
u           1.0359 0.2811 3.685 0.00106 **
v           0.9217 0.3787 2.434 0.02211 *
w           0.7261 0.3652 1.988 0.05744 .
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.625 on 26 degrees of freedom
Multiple R-squared: 0.4981,    Adjusted R-squared: 0.4402
F-statistic: 8.603 on 3 and 26 DF, p-value: 0.0003915
```

这个模型汇总显示了估计系数。它显示了关键的统计量，例如R^2和F统计量。它也显示了σ的一个估计、残差的标准误差。该汇总是如此重要，我们有一节全部用于理解汇总输出的内容（参见方法11.4）。

有专门的提取其他重要信息的函数，这些函数包括以下几种。

模型系数（点估计）

```
> coef(m)
(Intercept)         u         v         w
1.4222050 1.0358725 0.9217432 0.7260653
```

模型系数的置信区间

```
> confint(m)
                  2.5 %   97.5 %
(Intercept) -1.46302727 4.307437
```

```
u          0.45805053 1.613694
v          0.14332834 1.700158
w         -0.02466125 1.476792
```

模型残差

```
> resid(m)
          1           2           3           4           5           6
-1.41440465 1.55535335 -0.71853222 -2.22308948 -0.60201283 -0.96217874
          7           8           9          10          11          12
-1.52877080 0.12587924 -0.03313637 0.34017869 1.28200521 -0.90242817
         13          14          15          16          17          18
 2.04481731 1.13630451 -1.19766679 -0.60210494 1.79964497 1.25941264
         19          20          21          22          23          24
-2.03323530 1.40337142 -1.25605632 -0.84860707 -0.47307439 -0.76335244
         25          26          27          28          29          30
 2.16275214 1.53483492 1.65085364 -3.39647629 -0.46853750 3.12825629
```

残差平方和

```
> deviance(m)
[1] 68.69616
```

方差分析表

```
> anova(m)
Analysis of Variance Table

Response: y
          Df Sum Sq Mean Sq F value   Pr(>F)
u          1 27.916 27.9165 10.5658 0.003178 **
v          1 29.830 29.8299 11.2900 0.002416 **
w          1 10.442 10.4423  3.9522 0.057436 .
Residuals 26 68.696  2.6422
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

如果你发现把模型保存在一个变量中很麻烦，可以使用如下单行形式：

```
> summary(lm(y ~ u + v + w))
```

另请参阅

参见方法11.4。关于针对模型诊断的回归统计量的内容，请参见方法11.6。

11.4 理解回归的汇总结果

问题

创建了一个线性回归模型m。然而，你对从summary(m)中输出的结果感到疑惑。

讨论

模型汇总结果是很重要的，它让你接触到了最关键的回归统计量。以下是方法11.3的模型汇总：

```
> summary(m)

Call:
lm(formula = y ~ u + v + w)

Residuals:
    Min      1Q  Median      3Q     Max
-3.3965 -0.9472 -0.4708  1.3730  3.1283

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.4222     1.4036   1.013  0.32029
u             1.0359     0.2811   3.685  0.00106 **
v             0.9217     0.3787   2.434  0.02211 *
w             0.7261     0.3652   1.988  0.05744 .
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.625 on 26 degrees of freedom
Multiple R-squared: 0.4981,     Adjusted R-squared: 0.4402
F-statistic: 8.603 on 3 and 26 DF,  p-value: 0.0003915
```

让我们依照输出顺序逐个来分析这个汇总。我们从头到尾阅读它——即使最重要的统计量，*F*统计量，出现在最后。

调用

```
Call:
lm(formula = y ~ u + v + w)
```

当创建模型时，以上代码表明lm如何被调用，这对于把这个汇总放入适当的环境中是非常重要的。

残差统计量

```
Residuals:
    Min      1Q  Median      3Q     Max
-3.3965 -0.9472 -0.4708  1.3730  3.1283
```

理想的情况下，回归残差将有一个完美的正态分布。这些统计量帮助你从正态性中识别可能出现的偏差。普通最小二乘法在数学上保证产生均值为零的残差[注1]。因此中位数的符号表示偏斜的方向，而中位数的大小表示偏斜的程度。在这种情况下，中位数为负值，表明向左边偏斜。

如果残差有一个很好的钟形分布，第一个四分位数（1Q）和第三个四分位数（3Q）应该有大约相同的幅度。在这个例子中，与1Q相比，有较大幅度的3Q（1.3730比0.9472）表示我们的数据有轻微向右的偏斜，虽然负中位数使形状不那么明显。

最小和最大残差提供了一种快速的方法来检测数据中的极端离群值，因为极端离群值（在响应变量中）产生较大的残差。

系数

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.4222     1.4036   1.013  0.32029
u             1.0359     0.2811   3.685  0.00106 **
v             0.9217     0.3787   2.434  0.02211 *
w             0.7261     0.3652   1.988  0.05744 .
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

标记为Estimate的列包含由普通最小二乘法计算出的估计回归系数。

从理论上说，如果一个变量的系数是零，那么该变量是毫无意义的，它对模型毫无贡献。然而，这里显示的系数只是估计，它们不会正好为零。因此，我们不禁要问：从统计的角度而言，真正的系数是0的可能性有多大？这是t统计量和p值的目的，它们在汇总中分别标记为t value和Pr(>| t|)。

p值是一个概率。它估计系数不显著的可能性，所以越小越好。较大的p值是不理想的，因为它表明不显著的可能性很高。在这个例子中，系数u的p值仅为0.001 06，所以u可能是显著的。然而，w的p值是0.057 44；它刚好高于传统界限0.05，它表明w也可能是不显著的[注2]。有较大p值的变量是可以从模型中移除的候选变量。

R的一个方便的特点是它会标记显著性变量，帮助用户快速识别变量是否显著。注意列的右端包含双星号（**）、单星号（*）和点号（.）了吗？该列突出了显著性变量。底端标有“Signif. codes”的行给出了标记项的含义：

注1：　除非你运行无截距项的线性模型（参见方法11.5）。

注2：　显著性水平$\alpha = 0.05$是本书遵守的惯例。你的应用可能改为使用$\alpha = 0.10$、$\alpha = 0.01$，或其他一些值。有关这部分的内容，请参阅第9章。

***	*p*-value between 0 and 0.001
**	*p*-value between 0.001 and 0.01
*	*p*-value between 0.01 and 0.05
.	*p*-value between 0.05 and 0.1
(blank)	*p*-value between 0.1 and 1.0

标记为`Std.Error`的列是估计的回归系数的标准误差。标记为`t value`的列是t统计量，p值是据此来计算出来的。

残差标准误差

```
Residual standard error: 1.625 on 26 degrees of freedom
```

以上显示了残差的标准偏差，即ε的样本标准偏差。

R^2

```
Multiple R-squared: 0.4981,   Adjusted R-squared: 0.4402
```

R^2（判定系数）是衡量模型拟合质量的指标。它的值越大越好。在数学上，它是回归模型所能解释的响应变量y的方差比例。剩余的方差是模型不能解释的，它必须由其他因素解释（例如，未知变量或样本的变化）。在这种情况下，模型解释y的方差为0.4981（49.81%），剩余的0.5019（50.19%）未解释。

话虽如此，我强烈建议使用调整的R^2而不使用基本R^2。调整R^2考虑了模型中变量的数目，所以能实际地评估模型的有效性。在多元回归情况下，我会使用0.4402，而不是0.4981。

F统计量

```
F-statistic: 8.603 on 3 and 26 DF,  p-value: 0.0003915
```

F统计量告诉你模型是否显著。如果有任何回归系数为非零（即对某些i，$\beta_i \neq 0$），则该模型就是显著的。如果所有系数均为零（$\beta_1=\beta_2=...=\beta_n=0$），模型就是不显著的。

按照惯例，p值小于0.05表明该模型是显著的（一个或多个β_i为非零），然而p值超过0.05表明模型可能是不显著的。这里，模型是不显著的概率只有0.000 391。这个结果很理想。

大多数人先看R^2统计量。明智的统计学家先看F统计量，因为如果模型是不显著的，那么其他的都不重要。

另请参阅

更多关于从模型对象中提取统计量和相关信息的内容，请参见方法11.3。

11.5 运行无截距的线性回归

问题

需要运行一个线性回归，但你想设定截距为0。

解决方案

在回归公式右侧加上"+0"。这将使得lm以截距为0来设置模型：

```
> lm(y ~ x + 0)
```

相应的回归方程是：

$$y_i = \beta x_i + \varepsilon_i$$

讨论

线性回归通常包括截距项，所以它在R中是默认的。然而，极少数情况下，你可能会假设截距为零来拟合数据。在这种情况下，你做出一个模型假设：当x为0时，y应该是0。

当设定一个0截距时，lm的输出包括一个x的系数，但没有y的截距，如下所示。

```
> lm(y ~ x + 0)

Call:
lm(formula = y ~ x + 0)

Coefficients:
    x
2.582
```

我强烈建议你在继续进行分析之前，先检验模型的假设条件。运行一个有截距的回归方程；然后看看截距是否可以合理地设置为0。检验截距的置信区间。在这个例子中，置信区间为（−12.17，10.17）：

```
> confint(lm(y ~ x))
                 2.5 %    97.5 %
(Intercept) -12.171368 10.170070
x             2.012768  3.248262
```

因为置信区间包含零，所以从统计学来说，截距可能为零。现在，设定一个0截距后，应该重新运行回归分析。

11.6 运行有交户项的线性回归

问题

需要在你的回归中包含一个交互项。

解决方案

R语法可以指定回归公式的交互项。两个变量u和v的相互作用，通过在它们的名字间插入一个星号（*）来表示：

```
> lm(y ~ u*v)
```

这相当于指定模型$y_i = \beta_0 + \beta_1 u_i + \beta_2 v_i + \beta_3 u_i v_i + \varepsilon_i$，它包含一级交互项$\beta_3 u_i v_i$。

讨论

在回归中，当两个预测变量的乘积也是一个显著的预测变量时，交互作用就会出现。假设我们有两个预测变量u和v，并且要在回归中包含它们的交互作用。

可以用如下公式表示：

$$y_i = \beta_0 + \beta_1 u_i + \beta_2 v_i + \beta_3 u_i v_i + \varepsilon_i$$

这里，乘积$\beta_3 u_i v_i$项称为交互项。对于该方程式的R公式为：

```
y ~ u*v
```

当写入y~u*v时，R会自动地在模型中包括预测变量u、v以及它们的乘积。它基于这样的推理：如果一个模型包含交互项，例如，$\beta_3 u_i v_i$，那么回归理论告诉我们模型还应该包含交互项的组成变量u_i和v_i。

同样，如果有三个预测变量（u、v和w），并且要包括它们之间的所有交互作用，则由星号将它们分开：

```
y ~ u*v*w
```

这相当于回归方程：

$$y_i = \beta_0 + \beta_1 u_i + \beta_2 v_i + \beta_3 w_i + \beta_4 u_i v_i + \beta_5 u_i w_i + \beta_6 v_i w_i + \beta_7 u_i v_i w_i + \varepsilon_i$$

现在我们有了所有的一阶交互和二阶交互$\beta_7 u_i v_i w_i$。

然而，有时你可能不需要每个可能的交互。通过使用冒号（:），可以明确地指定一个单一乘积。例如，u:v:w表示乘积项$\beta u_i v_i w_i$而不是所有可能的交互作用。所以R的公式为：

```
y ~ u + v + w + u:v:w
```

这相当于回归方程：

$$y_i = \beta_0 + \beta_1 u_i + \beta_2 v_i + \beta_3 w_i + \beta_4 u_i v_i w_i + \varepsilon_i$$

冒号（:）是指纯乘法，而星号（*）是指乘法和所有乘法的构成项，这看似很奇怪。这是因为当包含交互时，通常包含它的组成项，这样把它们的默认值设为星号就合理了。

有一些额外的语法用来轻松地指定相互作用：

`(u + v + ... + w)^2`
　　包含所有变量(u, v, ..., w)和所有它们的一级交互。

`(u + v + ... + w)^3`
　　包含所有变量、所有它们的一级交互和二级交互。

`(u + v + ... + w)^4`
　　如上以此类推。

星号（*）和冒号（:）遵循“分配律”，所以下面的符号也是允许的：

`x*(u + v + ... + w)`
　　它等同于x*u+x*v+⋯+x*w（它也等同于x+u+v+...+w+x:u+x:v+...+x:w）。

`x:(u + v + ... + w)`
　　它等同于x:u + x:v + ... + x:w。

以上所有语法在你编写公式时给予一些灵活性。例如，以下三个公式是等价的：

```
y ~ u*v
y ~ u + v + u:v
y ~ (u + v)^2
```

它们都定义了相同的回归方程，$y_i = \beta_0 + \beta_1 u_i + \beta_2 v_i + \beta_3 u_i v_i + \varepsilon_i$。

另请参阅

公式的完整语法比这里所描述的更为丰富。详情请参阅《R in a Nutshell》（O'Reilly）或者《R Language Definition》。

11.7 选择最合适的回归变量

问题

正在创建一个新的回归模型，或者改善现有的模型。你有许多回归变量，而你需要选择这些变量中最有关系的那些。

解决方案

函数step可以执行逐步回归，前向或者后向。后向逐步回归开始时有许多变量，然后它移除无意义的变量：

```
> full.model <- lm(y ~ x1 + x2 + x3 + x4)
> reduced.model <- step(full.model, direction="backward")
```

前向逐步回归开始时有很少的变量，然后添加新的变量来改进模型，直到它不能再被改进：

```
> min.model <- lm(y ~ 1)
> fwd.model <- step(min.model, direction="forward", scope=( ~ x1 + x2 + x3 + x4 ))
```

讨论

当有很多的预测变量时，选择最好的子集会是相当困难的。添加和删除单个变量会影响整体，所以寻找最好的变量可能很冗长乏味。

函数step使搜索自动化。后向逐步回归是最简单的方法。开始时，它包括所有的预测变量。我们称它为全模型（*full model*）。模型汇总如下所示，它表明并非所有的预测变量都是统计显著的：

```
> full.model <- lm(y ~ x1 + x2 + x3 + x4)
> summary(full.model)

Call:
lm(formula = y ~ x1 + x2 + x3 + x4)

Residuals:
    Min      1Q Median     3Q    Max
-34.405 -5.153  2.025 6.525 19.186

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) 10.274      4.573   2.247   0.0337 *
x1         -102.298     47.558 -2.151   0.0413 *
```

```
x2          4.362     40.237  0.108    0.9145
x3         75.115     34.236  2.194    0.0377 *
x4         26.286     42.239  0.622    0.5394
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 12.19 on 25 degrees of freedom
Multiple R-squared: 0.8848,     Adjusted R-squared: 0.8664
F-statistic:     48 on 4 and 25 DF, p-value: 2.235e-11
```

我们要移除不显著的变量，所以我们调用step来逐步移除它们。移除了不显著变量后的模型称为简化模型（*reduced model*）：

```
> reduced.model <- step(full.model, direction="backward")
Start: AIC=154.58
y ~ x1 + x2 + x3 + x4

        Df Sum of Sq    RSS    AIC
- x2     1      1.75 3718.5 152.60
- x4     1     57.58 3774.3 153.04
<none>               3716.7 154.58
- x1     1    687.89 4404.6 157.68
- x3     1    715.67 4432.4 157.87

Step: AIC=152.6
y ~ x1 + x3 + x4

        Df Sum of Sq    RSS    AIC
- x4     1     77.82 3796.3 151.22
<none>               3718.5 152.60
- x3     1    737.39 4455.9 156.02
- x1     1    787.96 4506.4 156.36

Step: AIC=151.22
y ~ x1 + x3

        Df Sum of Sq    RSS    AIC
<none>               3796.3 151.22
- x1     1    884.44 4680.7 155.50
- x3     1    964.24 4760.5 156.01
```

step的输出显示所生成的模型序列。在这种情况下，step移除了x2和x4，只在最后的（简化）模型中保留了x1和x3。简化模型的汇总仅仅显示包含在模型中显著的预测变量：

```
> summary(reduced.model)

Call:
lm(formula = y ~ x1 + x3)
```

```
Residuals:
    Min      1Q  Median      3Q     Max
-36.192 -2.677   2.049   7.453  19.704

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  10.267      4.437   2.314   0.0285 *
x1          -79.265     31.604  -2.508   0.0185 *
x3           82.726     31.590   2.619   0.0143 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 11.86 on 27 degrees of freedom
Multiple R-squared: 0.8823,	Adjusted R-squared: 0.8736
F-statistic: 101.2 on 2 and 27 DF,  p-value: 2.842e-13
```

后向逐步回归很容易运行，但有时它开始时包含的所有变量是不可行的，因为有太多的候选变量。在这种情况下，使用前向逐步回归，它开始时不包含变量，然后逐步增加变量以优化回归。当模型不能再改进时，它便停止了。

开始时模型没有变量会看似很奇怪：

```
> min.model <- lm(y ~ 1)
```

这是一个有响应变量（y）但没有预测变量的模型（所有y的拟合值只是y的均值，如果没有预测变量，这种情况就是你会猜想到的）。

我们必须告诉step可以纳入模型的候选变量。这是参数scope的目的。scope是一个公式，其中，在波浪符号（~）左边没有变量而候选变量在右边：

```
> fwd.model <- step(min.model,
+                   direction="forward",
+                   scope=( ~ x1 + x2 + x3 + x4),
+                   trace=0 )
```

这里，我们看到x1、x2、x3和x4是纳入模型的候选变量（还包括trace=0来抑制step的大量输出）。结果模型只有两个显著的预测变量而没有给出不显著的预测变量：

```
> summary(fwd.model)

Call:
lm(formula = y ~ x3 + x1)

Residuals:
    Min      1Q  Median      3Q     Max
-36.192  -2.677   2.049   7.453  19.704
```

```
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  10.267      4.437   2.314   0.0285 *
x3           82.726     31.590   2.619   0.0143 *
x1          -79.265     31.604  -2.508   0.0185 *
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 11.86 on 27 degrees of freedom
Multiple R-squared: 0.8823,    Adjusted R-squared: 0.8736
F-statistic: 101.2 on 2 and 27 DF, p-value: 2.842e-13
```

通过包含x1和x3，排除x2和x4，前向算法和后向算法达成了相同的模式。这是一个示例性的例子，所以这并不令人惊讶。在实际应用中，建议尝试前向和后向两种回归方法，然后比较结果。你可能会对此感到惊讶。

最后，不要因为使用逐步回归而得意忘形。它也不是万能的，它不能把垃圾变成黄金，而且它绝对不能替代小心明智地选择预测变量的过程。你可能会想："噢天！我可以为我的模型产生每个可能的相互作用，然后让step选择最好的！我会得到合适的模型！"你所想的如下所示。

```
> full.model <- lm(y ~ (x1 + x2 + x3 + x4)^4)          # All possible interactions
> reduced.model <- step(full.model, direction="backward")
```

这不会奏效：大部分的交互项是没有意义的。函数step变得不堪重负，而你留下许多不显著的项。

另请参阅

参见方法11.24。

11.8 对数据子集回归

问题

要对一部分数据（不是整个数据集）拟合一个线性模型。

解决方案

函数lm有一个参数subset，它指定了哪些数据元素应该用于拟合。该参数的值可以是任何用于索引数据的表达式。下面显示了只用前100个观察值的回归拟合：

```
> lm(y ~ x, subset=1:100)              # Use only x[1:100]
```

讨论

你会经常需要只回归数据中的子集。例如，当使用样本内数据创建模型和样本外数据来检验它时，这种情况就会发生。

函数lm有一个参数subset，它选择用于拟合的观察值。subset的值是一个向量。它可以是一个索引值向量，在这种情况下，lm只从数据中选择指定的观察值。它也可以是一个逻辑向量，长度与数据相同，在这种情况下，lm选择有相应TRUE值的观察值。

假设你有（x, y）对的1000个观察值，并希望通过使用这些观察值的前一半来拟合你的模型。可以设置参数subset的值为1：500，它表明lm应该使用1～500的观察值：

```
> lm(y ~ x, subset=1:500)
```

更普遍地，可以调用表达式1:floor(length(x)/2)来选择数据的前半段，无论它的大小是多少：

```
> lm(y ~ x, subset=1:floor(length(x)/2))
```

比方说，数据是从几个实验室收集而来，有一个因子lab，它标识数据所来源的实验室。通过使用一个只对某些观察值为TRUE的逻辑向量，你可以把回归限制在仅仅在新泽西州收集的观察值上：

```
> lm(y ~ x, subset=(lab == "NJ"))
```

11.9 在回归公式中使用表达式

问题

要对计算出的值进行回归，而不是简单的变量，但回归公式的语法似乎禁止这样做。

解决方案

在I(...)运算符中嵌入已计算值的表达式。这将强制R计算这个表达式并使用计算值进行回归。

讨论

如果你要对u和v的和进行回归，那么以下是回归方程：

$$y_i = \beta_0 + \beta_1(u_i + v_i) + \varepsilon_i$$

你要如何把上面的方程写成一个回归公式呢？以下代码不能做到：

```
> lm(y ~ u + v)                # Not quite right
```

这里，R把u和v解释为两个分开的预测变量，每个有它自己的回归系数。同样，假设回归方程是：

$$y_i = \beta_0 + \beta_1 u_i + \beta_2 u_i^2 + \varepsilon_i$$

以下代码不能起作用：

```
> lm(y ~ u + u^2)              # That's an interaction, not a quadratic term
```

R将把u^2解释为一个交互项（参见方法11.6），而不是u的平方。

解决方案使用`I(...)`运算符，把你要回归的表达式作为该运算符的参数，从而禁止表达式解释为回归公式。相反，它迫使R来计算该表达式的值，然后直接把这个值纳入回归。因此第一个例子可以表示如下：

```
> lm(y ~ I(u + v))
```

作为对这个命令的响应，R先计算$u+v$，然后用计算出的和对变量y进行回归。

对第二个例子，正确的表示如下：

```
> lm(y ~ u + I(u^2))
```

这里，R先计算u的平方，然后用总和 $u+u^2$ 对y进行回归。

在回归公式中，所有基本二元运算符（+、-、*、/、^）都有特殊的含义。出于这个原因，当需要把计算值纳入回归时，必须使用`I(...)`运算符。

这些嵌入式转换的奇妙之处在于R记得这些转换，并在你用模型进行预测时应用这些转换。考虑第二个例子描述的二次模型。它使用u和u^2，但我们只提供u的值而让R做繁重的工作。我们不需要自己来计算u的平方：

```
> m <- lm(y ~ u + I(u^2))
> predict(m, newdata=data.frame(u=13.4))    # R plugs u and u^2 into the calculation
       1
11531.59
```

另请参阅

参见方法11.10关于多项式回归的特殊情况。参见方法11.11把其他数据转换纳入回归。

11.10 多项式回归

问题

需要用x的多项式对y进行回归。

解决方案

在回归公式中调用函数poly(x,n)来对x的一个n阶多项式进行回归。这个例子把y建模为x的三次函数：

```
> lm(y ~ poly(x,3,raw=TRUE))
```

该例子的公式相当于以下三次回归公式：

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x^2_i + \beta_3 x_i^3 + \varepsilon_i$$

讨论

当我第一次在R中使用多项式模型时，我输入了如下复杂的代码：

```
> x_sq <- x^2
> x_cub <- x^3
> m <- lm(y ~ x + x_sq + x_cub)
```

显而易见，这很令人讨厌，这里的额外变量使我的工作空间凌乱。

下面的写法就容易得多了：

```
> m <- lm(y ~ poly(x,3,raw=TRUE))
```

设定raw=TRUE是必要的。没有它，函数poly计算正交多项式而不是简单的多项式。

以上写法除了方便外，一个巨大的优势在于当你从该模型做出预测时，R将计算x的所有乘方项（参见方法11.18）；否则，每次使用模型时，你要首先自己计算出x^2和x^3。

另一个应用函数poly的理由是，你不能用以下方式书写回归公式：

```
> lm(y ~ x + x^2 + x^3)         # Does not do what you think!
```

R将x^2和x^3解释为交互项，而不是x的乘方。由此产生的模型是一个单项线性回归，完全不符合你的期望。你可以如下书写回归公式：

```
> lm(y ~ x + I(x^2) + I(x^3))
```

但这会变得很冗余。请调用poly。

另请参阅

更多关于交互项的内容，请参见方法11.6。参见方法11.11来查阅其他对于回归数据的转换。

11.11 转换数据的回归

问题

要为x和y构建一个回归模型，但它们没有线性关系。

解决方案

可以在回归公式中嵌入需要的转换。例如，如果y必须转换为$\log(y)$，那么回归公式变为：

```
> lm(log(y) ~ x)
```

讨论

回归函数lm的一个关键假设是变量有线性关系。在一定程度上，如果这种假设不成立，由此产生的回归就会毫无意义。

幸运的是，许多数据集在应用lm前可以转换为线性关系。图11-1显示了一个指数衰减的例子。图11-1a显示了原始数据z。虚线显示了原始数据的线性回归；很明显，这是一个糟糕的拟合。如果数据真是指数关系，那么一个可能的模型是：

$$z = \exp[\beta_0 + \beta_1 t + \varepsilon]$$

其中，t是时间，exp[.]是指数函数（e^x）。这当然是非线性的，但我们可以通过对它取对数而使它线性化：

$$\log(z) = \beta_0 + \beta_1 t + \varepsilon$$

在R中，这种回归是很简单的，因为我们可以把对数转换直接嵌入回归公式：

```
> m <- lm(log(z) ~ t)
> summary(m)

Call:
lm(formula = log(z) ~ t)
```

```
Residuals:
     Min       1Q   Median       3Q      Max
-0.17366 -0.09070 -0.03761  0.08080  0.36454

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.22059    0.01653  -13.35   <2e-16 ***
t           -1.16110    0.02056  -56.46   <2e-16 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1275 on 98 degrees of freedom
Multiple R-squared: 0.9702, Adjusted R-squared: 0.9699
F-statistic: 3188 on 1 and 98 DF, p-value: < 2.2e-16
```

图11-1b中显示log(z)随时间变化的图形。叠加在散点图上的是它们的回归线。拟合似乎要好得多；这也可以由回归输出结果证实：R^2= 0.97，而对原始数据的线性回归的R^2=0.76。

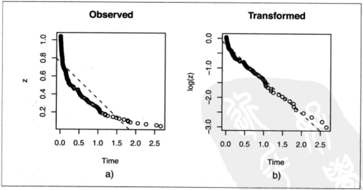


图11-1：转换数据为线性关系

可以把其他函数嵌入公式。如果你认为关系是二次的，那么你可以使用平方根变换：

```
> lm(sqrt(y) ~ month)
```

当然也可以对公式两边的变量应用变换。以下公式在x平方根上回归y：

```
> lm(y ~ sqrt(x))
```

以下是对x的对数和y的对数的回归：

```
> lm(log(y) ~ log(x))
```

另请参阅

参见方法11.12。

11.12 寻找最佳幂变换

问题

需要通过对响应变量应用一个幂变换来改进你的线性模型。

解决方案

使用Box-Cox程序，它由软件包MASS中的函数boxcox执行。这个程序会确定一个指数λ，这样把y转换为y^λ会改进你的模型拟合：

```
> library(MASS)
> m <- lm(y ~ x)
> boxcox(m)
```

讨论

为了说明Box-Cox程序，使用方程式$y^{-1.5}=x+\varepsilon$创建一些人工数据，其中ε是一个误差项：

```
> x <- 10:100
> eps <- rnorm(length(x), sd=5)
> y <- (x + eps)^(-1/1.5)
```

随后我们会（错误地）用一个简单线性回归对数据建模，并得到一个调整的R^2，0.6935：

```
> m <- lm(y ~ x)
> summary(m)

Call:
lm(formula = y ~ x)

Residuals:
      Min        1Q    Median        3Q       Max
-0.038866 -0.016128 -0.004948  0.008367  0.083354

Coefficients:
             Estimate   Std. Error t value Pr(>|t|)
(Intercept) 1.580e-01  5.710e-03   27.67   <2e-16 ***
x          -1.340e-03  9.368e-05  -14.30   <2e-16 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 0.02347 on 89 degrees of freedom
Multiple R-squared: 0.6969,    Adjusted R-squared: 0.6935
F-statistic: 204.6 on 1 and 89 DF, p-value: < 2.2e-16
```

当绘制残差和拟合值的图时，我们看出这里的回归是有错误的：

```
> plot(m, which=1)            # Plot only the fitted vs residuals
```

图11-2a是绘制的残差和拟合值的图，它是抛物线的形状。一个可能的改变是对y进行幂变换，所以我们运行Box-Cox程序：

```
> library(MASS)
> bc <- boxcox(m)
```

函数boxcox创建一个图，如图11-2b所示，它绘制了λ的值和相应结果模型的对数似然值。我们要最大化这个对数似然值，所以该函数绘制一条最佳值的线，并画线表示它的置信区间的范围。在这种情况下，最佳值大约是-1.5，它的置信区间约为$(-1.75, -1.25)$。

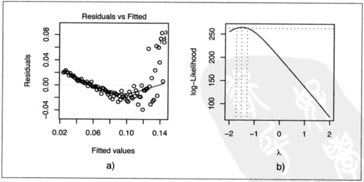


图11-2：Box-Cox图

奇怪的是，函数boxcox不返回λ的最佳值。相反，它返回显示在图形中的(x, y)对。找到λ值是很容易的，它产生y的最大对数似然估计。我们调用函数which.max：

```
> which.max(bc$y)
[1] 13
```

然后它给出相应的λ的位置：

```
> lambda <- bc$x[which.max(bc$y)]
> lambda
[1] -1.515152
```

函数显示λ的最佳值为−1.515。在实际应用中，我会强烈建议你解释这个数值，并选择有意义的指数——而不是盲目地接受这个"最佳"值。可以使用图来协助你对这个数值的解释。这里，我们就采用R给出的−1.515。

我们可以对y应用这个幂变换，然后拟合修正的模型，它给出了一个更理想的R^2值，0.9595：

```
> z <- y^lambda
> m2 <- lm(z ~ x)
> summary(m2)

Call:
lm(formula = z ~ x)

Residuals:
     Min       1Q   Median       3Q      Max
-12.4325  -3.6608  -0.8793   2.3856  15.0977

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.61801    1.36496   0.453    0.652
x            1.03382    0.02239  46.164   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.612 on 89 degrees of freedom
Multiple R-squared: 0.9599,     Adjusted R-squared: 0.9595
F-statistic: 2131 on 1 and 89 DF,  p-value: < 2.2e-16
```

对于那些更喜欢单行模式的人，该变换可以直接嵌入到修正的回归公式中：

```
> m2 <- lm(I(y^lambda) ~ x)
```

默认情况下，boxcox在范围−2～+2内搜索λ值。你可以通过参数lambda改变搜索范围。详情参见帮助说明页。

我建议把Box-Cox给出的结果作为一个起始点，而不是一个确切的答案。如果λ的置信区间包括1.0，那么可能没有幂变换是实际有用的。通常，检查变换前的残差和变换后的残差，它们是否真的改进了？

另请参阅

参见方法11.11和方法11.15。

11.13 回归系数的置信区间

问题

正在运行线性回归，需要回归系数的置信区间。

解决方案

把回归模型保存在一个对象中，然后调用函数confint以提取置信区间：

```
> m <- lm(y ~ x1 + x2)
> confint(m)
```

讨论

解决方案使用模型$y = \beta_0 + \beta_1(x_1)_i + \beta_2(x_2)_i + \varepsilon_i$。函数confint返回截距（即$\beta_0$）的置信区间、$x_1$的系数（即$\beta_1$）以及$x_2$的系数（即$\beta_2$）：

```
> confint(m)
                2.5 %    97.5 %
(Intercept) -5.332195 12.361525
x1          -3.009250  1.282295
x2           1.892837  7.093099
```

默认情况下，confint使用置信水平95%。使用参数level选择一个不同的置信水平：

```
> confint(m, level=0.99)
                 0.5 %    99.5 %
(Intercept) -8.4316652 15.460995
x1          -3.7610142  2.034059
x2           0.9818897  8.004046
```

另请参阅

软件包arm的函数coefplot可以绘制回归系数的置信区间。

11.14 绘制回归残差

问题

需要你的回归残差的一个可视化显示。

解决方案

你可以从得到的图形中选择残差图来绘制模型对象：

```
> m <- lm(y ~ x)
> plot(m, which=1)
```

讨论

通常情况下，绘制一个回归模型对象会产生多个诊断图。你可以通过指定`which=1`来选择残差图。

图11-3显示了从方法11.1得到的残差图。R绘制一条通过残差的平滑线，把它作为发现显著模式的可视化工具——例如，斜率或抛物线形。

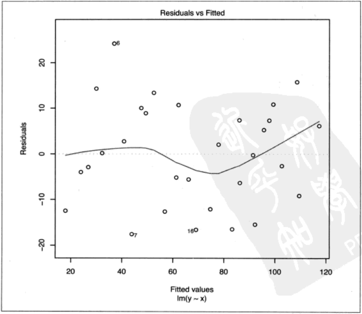


图11-3：绘制回归残差

另请参阅

参见方法11.15，它包含了残差图和其他诊断图的例子。

11.15 诊断线性回归

问题

运行了一个线性回归。现在需要通过诊断检验来验证模型的质量。

解决方案

首先，绘制模型对象，它会产生几个诊断图：

```
> m <- lm(y ~ x)
> plot(m)
```

然后，通过查看残差的诊断图或者调用软件包car中的函数outlier.test来识别可能的离群值：

```
> library(car)
> outlier.test(m)
```

最后，识别任何过度影响的观察值（参见方法11.16）。

讨论

R使你觉得线性回归是很容易的：只要调用函数lm。然而，拟合数据仅仅是一个开始。你的任务是确定拟合模型是否奏效并是否能达到很好的效果。

在进行其他步骤之前，你必须有统计上显著的模型。检验模型汇总中的*F*统计量（参见方法11.4），并确定*p*值对你的目的而言是足够小的。按照惯例，它应该小于0.05，否则你的模型可能是毫无意义的。

简单地绘制模型对象产生一些有用的诊断图：

```
> m <- lm(y ~ x)
> plot(m)
```

图11-4显示了一个很好的回归诊断图：

- 在残差－拟合图中的点没有特别的模式，大多数呈随机分布。
- 在正态Q-Q图中的点基本落在线上，这表明残差服从正态分布。

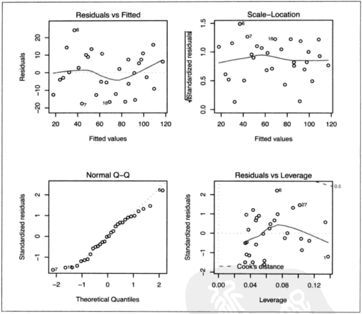


图11-4：理想拟合的诊断图

- 在大小-位置图和残差-影响图中，点以小组形式存在并且离中心不远。

相比之下，图11-5显示了一个不那么好的回归诊断。观察到残差-拟合图有一个明确的抛物线形。这就告诉我们，模型是不完整的：它缺少一个二次因子，该因子可以解释y的更多变化。残差的其他模式说明有额外的问题：例如，一个圆锥形状可能表明y有非齐性方差。解释这些模式是一门艺术，所以我建议在评估残差图形时，查看有关线性回归的好的参考书。

不那么好的诊断还有其他的问题。例如，正态Q-Q图比理想的回归图有更多离开线的点；大小－位置图和残差－影响图都显示离开中心的离散点，这表明某些点对回归有过多的影响。

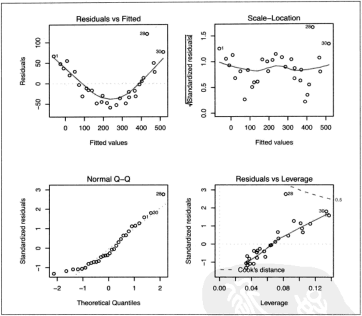


图11-5：拟合不理想的诊断图

另一种模式是28号点在每个图形都显得突出。这提醒我们这个观察值有些异样。例如，该点可能是离群值。我们可以调用软件包car中的函数outlier.test检验它：

```
> outlier.test(m)

max|rstudent| = 3.183304, degrees of freedom = 27,
unadjusted p = 0.003648903, Bonferroni p = 0.1094671

Observation: 28
```

outlier.test识别出了模型大多数的离群观察值。在这个例子中，它识别出了观察值28并且确定它是一个离群值。

另请参阅

参见方法11.4和方法11.16。软件包car不是R的标准发布版的一部分；参见方法3.9。

11.16 识别有影响的观察值

问题

需要识别对回归模型有最大影响的观察值。这对诊断数据中可能存在的问题是十分有用的。

解决方案

函数influence.measures包含了几个有用的统计量以识别有影响的观察值，它用一个星号（*）标记有显著影响的观察值。它主要的参数是回归所得到的模型对象：

```
> influence.measures(m)
```

讨论

这个方法的标题可以是“识别有过度影响的观察值”，但这可能是多余的。所有观察值都影响回归模型，哪怕只有一点儿影响。当统计学家说一个观察值是有影响的（influential）时，这意味着移除这个观察值很大程度上会改变所拟合的回归模型。我们需要识别这些观察值，因为它们可能是扭曲模型的离群值。我们应该自己调查这些值。

函数influence.measures报告了几个统计量：DFBETAS、DFFITS、协方差比，库克（Cook）距离以及帽子矩阵的值。如果这些度量值中的任何一个表明一个观察值是有影响的，则该函数会在该观察值的右端用一个星号（*）标记：

```
> influence.measures(m)
Influence measures of
       lm(formula = z ~ x) :

    dfb.1_     dfb.x  dffit cov.r   cook.d    hat inf
1  0.64663 -0.56267 0.6466  1.04 1.98e-01 0.1373
2  0.40133 -0.33798 0.4020  1.11 8.01e-02 0.1137
3  0.29023 -0.23992 0.2914  1.14 4.29e-02 0.1035
4  0.13917 -0.11036 0.1409  1.16 1.02e-02 0.0862
5  0.40997 -0.33125 0.4133  1.05 8.34e-02 0.0932
.
. (etc.)
.
25 -0.00985  0.02449 0.0338  1.16 5.93e-04 0.0700
26 -0.02698  0.06341 0.0856  1.15 3.78e-03 0.0740
27 -0.16813  0.31877 0.3888  1.09 7.47e-02 0.1017
28 -0.34708  0.74122 0.9580  0.62 3.46e-01 0.0830 *
29 -0.09644  0.18050 0.2187  1.17 2.44e-02 0.1045
30 -0.37679  0.64112 0.7398  0.97 2.51e-01 0.1339
```

这是方法11.15的模型，我们怀疑28号观察值是一个离群值。有一个星号标记了该观察值，确认它是过度影响值。

这个方法可以识别有影响的观察值，但你不应该习惯性地删除它们。这里需要一些判断。这些观察值是改善了你的模型还是破坏了它呢？

另请参阅

参见方法11.15。使用help(influence.measures)以得到一列有影响的度量值和一些相关函数。参阅一本关于回归的教科书，了解各种影响度量值的问题。

11.17 残差自相关检验

问题

运行了一个线性回归并需要检验残差的自相关性。

解决方案

Durbin–Watson检验可以检验残差的自相关性。该检验由软件包lmtest中的函数dwtest执行：

```
> library(lmtest)
> m <- lm(y ~ x)                # Create a model object
> dwtest(m)                     # Test the model residuals
```

输出包括一个p值。按照惯例，如果$p<0.05$，则残差显著相关；如果$p>0.05$，则没有提供它们相关的证据。

可以通过绘制残差的自相关函数（ACF）来对自相关性执行一个可视化的检验：

```
> acf(m)                        # Plot the ACF of the model residuals
```

讨论

Durbin–Watson检验通常用于时间序列分析，但它最初的创建是用来在回归残差中诊断自相关。我们是不希望看到残差中有自相关的，因为它扭曲了回归统计量，例如F统计量和回归系数的t统计量。自相关的存在表明模型缺少一个有用的预测变量，或者它应该包括一个时间序列的组成部分，如趋势或季节性的指标。

第一个例子建立了一个简单的回归模型，然后检验残差的自相关。该检验返回了一个p值0.077 55，这表明不存在显著的自相关：

```
> m <- lm(y ~ x)
> dwtest(m)

        Durbin-Watson test

data: m
DW = 1.5654, p-value = 0.07755
alternative hypothesis: true autocorrelation is greater than 0
```

第二个例子显示残差存在自相关。*p*值只有0.012 98，所以很可能存在自相关：

```
> m <- lm(y ~ z)
> dwtest(m)

        Durbin-Watson test

data: m
DW = 1.2946, p-value = 0.01298
alternative hypothesis: true autocorrelation is greater than 0
```

默认情况下，dwtest执行单边检验，并回答这个问题：残差的自相关大于零吗？如果模型可以出现负自相关（这是一种可能的情况），那么你应该通过设置alternative参数来执行双边检验：

```
> dwtest(m, alternative="two.sided")
```

Durbin–Watson检验也由软件包car中的函数durbin.watson执行。我建议只使用函数dwtest，因为我认为它的输出更有易读性。

另请参阅

R的标准发布版中既没有软件包lmtestest，也没有软件包car；参见方法3.6和方法3.9。更多关于自相关检验的内容，请参见方法14.13和方法14.16。

11.18 预测新值

问题

需要从回归模型中预测新值。

解决方案

把预测变量数据保存为一个数据框。调用函数predict，将数据框的值设置为参数newdata的值：

```
> m <- lm(y ~ u + v + w)
> preds <- data.frame(u=3.1, v=4.0, w=5.5)
> predict(m, newdata=preds)
```

讨论

一旦有了一个线性模型，做出预测是很容易的，因为函数predict执行所有繁重的任务。唯一的烦恼是安排一个数据框来包含数据。

函数predict返回一个预测值向量，每个预测对应于数据的一行。在解决方案中的例子包含一行数据，所以predict返回一个值：

```
> preds <- data.frame(u=3.1, v=4.0, w=5.5)
> predict(m, newdata=preds)
       1
12.31374
```

如果你的预测变量数据包含多行，则每行都应有一个相应的预测值：

```
> preds <- data.frame(
+               u=c(3.0, 3.1, 3.2, 3.3),
+               v=c(3.9, 4.0, 4.1, 4.2),
+               w=c(5.3, 5.5, 5.7, 5.9) )
> predict(m, newdata=preds)
       1        2        3        4
11.97277 12.31374 12.65472 12.99569
```

这里要指出的是，新数据不需要包含响应变量的值，只有预测变量。毕竟，你正在试图计算响应变量，因此希望你提供它对R而言是不合理的。

另请参阅

本方法给出的只是预测变量的点估计。关于预测值的置信区间，请参阅方法11.19。

11.19 建立预测区间

问题

正在使用线性回归模型做预测。你想知道预测值的区间，即预测值的分布范围。

解决方案

调用函数predict并设置参数interval="prediction"：

```
> predict(m, newdata=preds, interval="prediction")
```

讨论

这是方法11.18的延续。首先，把预测变量数据打包成一个数据框，然后调用函数predict来进行预测。这里，增加参数interval="prediction"以获得预测区间。

下面是方法11.18的例子，现在给出了预测区间。新的lwr和upr列分别是区间的上限和下限：

```
> predict(m, newdata=preds, interval="prediction")
       fit      lwr      upr
1 11.97277 7.999848 15.94569
2 12.31374 8.272327 16.35516
3 12.65472 8.540045 16.76939
4 12.99569 8.803251 17.18813
```

默认情况下，predict使用了0.95的置信水平。你可以通过level参数来改变它。

注意，这些预测区间对偏离正态是极其敏感的。如果你怀疑响应变量不是正态分布的，那么考虑使用非参数方法，例如自助法（参见方法13.8）来计算预测区间。

11.20 运行单因素方差分析

问题

将数据分成几组，每组数据都是正态分布的。你想要知道这些组是否有显著不同的均值。

解决方案

使用一个因子来定义组，然后应用函数oneway.test：

```
> oneway.test(x ~ f)
```

这里，x是一个数值向量， f是定义组的一个因子。输出包括一个p值。按照惯例，$p<0.05$表明两个或多个组有显著不同的均值，然而，$p>0.05$没有提供这样的证据。

讨论

比较组的均值是一项普通的任务。单因素方差分析执行这个比较，并计算它们统计上相同的概率。较小的p值表示两个或多个组可能有不同的均值。（它不表明所有组有不同的均值。）

基本的方差分析检验假设数据有一个正态分布，或者至少它是非常接近的钟形的。如果不是，使用Kruskal-Wallis检验（参见方法11.23）。

我们可以以股票市场的历史数据为例来说明方差分析。股票市场在某些月份里比其他月份有更多盈利吗？例如，一个普遍的民间谣传说，10月份对股市投资者是糟糕的一个月[注3]。我通过创建一个向量和一个因子来探讨这个问题。向量r包含标准普尔500种股票指数的日常变动百分率，它是股票市场绩效的广泛测量。因子mon表示该变动发生的日历月：1月、2月、3月等。数据是取自1950—2009年。

单因素方差分析显示的*p*值为0.033 47：

```
> oneway.test(r ~ mon)

	One-way analysis of means (not assuming equal variances)

data: r and mon
F = 1.9102, num df = 11.000, denom df = 5930.142, p-value = 0.03347
```

我们可以得出结论，股市会根据日历月份发生重大变化。

但是，在你跑到你的经纪人前翻看你的投资组合时，你要检查一些事：模式在最近有没有改变？你可以通过指定参数subset来限制分析最近的数据。这对oneway.test和函数lm都奏效。subset包含所要分析的观察值的索引；其他所有的观察值都忽略了。这里，我们把2500个最新观察值赋予索引，这大约是10年的数据：

```
> oneway.test(r ~ mon, subset=tail(seq_along(r),2500))

	One-way analysis of means (not assuming equal variances)

data: r and mon
F = 0.7973, num df = 11.000, denom df = 975.963, p-value = 0.643
```

在过去10年中，这些月间的差异消失了。较大的*p*值为0.643，它表示股票的变动率没有根据不同的日历月进行显著的变化。显然，那些显著不同是过去的事情了。

注意，oneway.test的输出里有文字：“（not assuming equal variances）”（不假定方差相等）。如果你知道这些组有相等的方差，你会设定参数var.equal=TRUE得到不保守的检验：

```
> oneway.test(x ~ f, var.equal=TRUE)
```

也可以通过调用函数aov来执行单因素方差分析，如下所示。

```
> m <- aov(x ~ f)
> summary(m)
```

然而，函数aov通常假设相等的方差，所以它比oneway.test更加死板。

注3：　马克·吐温说过：“10月，这对于股票投机是一个特别危险的月份。其他的是7月、1月、9月、4月、11月、5月、3月、6月、12月、8月和2月。”

另请参阅

如果均值有显著不同，使用方法11.22查看实际偏差。如果数据不是方差分析所要求的正态分布，使用方法11.23。

11.21 创建交互关系图

问题

多因素方差分析：应用两个或多个分类变量作为预测变量的方差分析。需要得到预测变量之间可能的交互关系的可视化检验。

解决方案

调用函数interaction.plot：

```
> interaction.plot(pred1, pred2, resp)
```

这里，pred1和pred2是两个分类预测变量，而resp是响应变量。

讨论

方差分析是线性回归的一种形式，因此理想的情况是每一个预测变量和响应变量间有线性关系。非线性的来源之一是两个预测变量间的交互关系：当一个预测变量的值发生变化时，其他预测变量和响应变量的关系也随之发生变化。检验预测变量之间的相互作用是一个基本的诊断。

软件包faraway中包含一个名为rats的数据集。在这个数据集中，treat和poison是分类变量，而time是响应变量。当绘制poison对time的图形时，我们寻找能够表明线性关系的直线、平行线。然而，调用函数interaction.plot得到的图形显示存在交互关系：

```
> library(faraway)
> data(rats)
> interaction.plot(rats$poison, rats$treat, rats$time)
```

结果图形如图11-6所示。每条线绘制了time和poison的关系。线之间的不同在于每条线代表不同的treat值。各条线应该是平行的，而前两条线不完全平行。由此可见，改变treat的值“扭曲”了线条，在poison和time的关系中引入了非线性关系。

这预示着我们应该检验可能存在的交互关系。对于这个数据，它只是恰巧有一个交互关系，而在统计上并不显著。这里的经验是：可视化检验是有用的，但它并非万无一失。随后应该跟进一个统计检查。

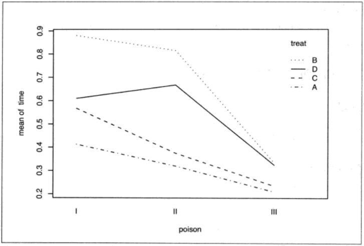


图11-6：treat和poison交互关系图

另请参阅

参见方法11.6。

11.22 找到组间均值的不同

问题

将数据分成组，方差分析显示这些组有显著不同的均值。你需要知道所有组的均值的差异。

解决方案

调用函数aov运行方差分析检验，它返回一个模型对象。然后对这个模型对象应用函数TukeyHSD：

```
> m <- aov(x ~ f)
> TukeyHSD(m)
```

这里，x是数据，f是分组因子。你可以绘制TukeyHSD的结果以得到这个差异的图形展示：

```
> plot(TukeyHSD(m))
```

讨论

方差分析检验是很重要的，因为它能告诉你组间均值是否有差异。但这个检验不能识别哪个组是不同的，而且它也不能显示它们的差异。

函数TukeyHSD可以计算这些差异，并且帮助你识别最大的均值。它使用由John Tukey发明的方法“诚实的显著性差异”。

通过继续探讨方法11.20中的例子来说明函数TukeyHSD，在方法11.20中把每日的股票市场变动按照月份分组。这里，我们将根据工作日来分组，调用函数wday确定一周发生变化的天（周一，…，周五）。我使用前2500个观察值，它们大致涵盖从1950—1960年的股票市场变化率数据：

```
> oneway.test(r ~ wday, subset=1:2500)

	One-way analysis of means (not assuming equal variances)

data: r and wday
F = 12.6051, num df = 4.000, denom df = 1242.499, p-value = 4.672e-10
```

p值接近于零，这表明不同工作日的股票市场平均变动是显著不同的。为了调用函数TukeyHSD，我首先调用aov函数执行方差分析检验，它返回一个模型对象，然后对该对象应用函数TukeyHSD：

```
> m <- aov(r ~ wday, subset=1:2500)
> TukeyHSD(m)
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = r ~ wday, subset = 1:2500)

$wday
               diff          lwr       upr     p adj
Tue-Mon  0.13131281  0.007025859 0.2555998 0.0323054
Wed-Mon  0.23846814  0.114846431 0.3620898 0.0000015
Thu-Mon  0.22371916  0.099432205 0.3480061 0.0000094
Fri-Mon  0.31728974  0.192816883 0.4417626 0.0000000
Wed-Tue  0.10715532 -0.015961497 0.2302721 0.1222470
Thu-Tue  0.09240635 -0.031378436 0.2161911 0.2481996
Fri-Tue  0.18597693  0.062005489 0.3099484 0.0004185
Thu-Wed -0.01474898 -0.137865796 0.1083678 0.9975329
Fri-Wed  0.07882161 -0.044482882 0.2021261 0.4064721
Fri-Thu  0.09357058 -0.030400857 0.2175420 0.2378246
```

在输出表中的每一行代表两组均值的差异（diff）以及该差异的置信区间的上限和下限（lwr和upr）。例如，表中的第一行比较Tue（星期二）组和Wed（星期三）组：它们均值的差异是0.1313，差异的置信区间为（0.0070，0.2556）。

浏览这张表，我们可以看到Fri～Mon（周五～周一）间的比较有最大差异，差异值是0.3173。

TukeyHSD一个很有用的功能在于，它可以直观地显示这些差异。简单地绘制函数返回值的图形：

```
> plot(TukeyHSD(m))
```

图11-7显示了股市例子的图。水平线绘制出了每对比较的置信区间。通过这个可视化的图形，你可以迅速地看到几个置信区间跨越零值，这表明差异不一定是显著的。你还可以看到，Fri～Mon（周五～周一）对有最大的差异，因为它们的置信区间远远地偏向右边。

另请参阅

参见方法11.20。

11.23 执行稳健方差分析

问题

将数据分成几组。各组数据不是正态分布的，但它们的分布有相似的形状。你需要执行一个类似方差分析的检验——你想知道这些组的中位数是否显著不同。

解决方案

创建一个因子用于定义你的数据组。调用函数kruskal.test，它实施Kruskal-Wallis检验。不同于方差分析检验，这个检验不依靠数据的正态性：

```
> kruskal.test(x ~ f)
```

这里，x是数据向量，f是一个分组因子。输出结果包含一个p值。依照惯例，$p < 0.05$表示在两个或更多组的中位数之间有显著差异，然而$p > 0.05$不提供这样的证明。

讨论

通常，方差分析假设数据是正态分布的。它可以允许数据些许偏离正态，但极度偏离正态会产生无意义的p值。

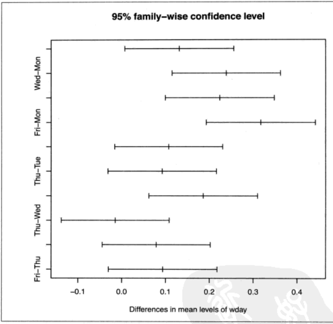


图11-7：组间均值的差异

Kruskal-Wallis检验是方差分析的一个非参数版本，这意味着它不假定正态性。然而，它假设各组有相同形状的分布。只要数据是非正态分布或者未知分布的，你就应该使用Kruskal-Wallis检验。

它的零假设是，所有数据组具有相同的中位数。拒绝原假设（$p<0.05$）并不表示所有组是不同的，但它确实建议两个或更多的组是不同的。

有一年，我教授94名本科学生“商务统计”课程。该课程有一个期中考试，考试前有四个作业。我想知道：完成作业和在考试中发挥良好之间的关系？如果这两者之间没有关系，那么作业是无关紧要的，需要反思作业的必要性。

我创建了一个成绩向量，每个向量对应一名学生。我还创建了一个平行因子，它包含学生完成作业的数量：

```
> print(midterm)
[1] 0.8182 0.6818 0.5114 0.6705 0.6818 0.9545 0.9091 0.9659 1.0000 0.7273
[11] 0.7727 0.7273 0.6364 0.6250 0.6932 0.6932 0.0000 0.8750 0.8068 0.9432
.
. (etc.)
.
[91] 0.9006 0.7029 0.9171 0.9628
> print(hw)
 [1] 4 4 2 3 4 4 4 3 4 4 3 3 1 4 4 3 3 4 3 4 4 4 3 2 4 4 4 4 3 2 3 4 4 0 4 4 4 4
[39] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 2 4 4 4 4 4 1 4 4 4 4 4 4 4 4 4 4 2 4 4
[77] 3 4 2 2 4 4 4 4 1 3 0 4 4 0 4 3 4 4
Levels: 0 1 2 3 4
```

注意，变量hw（虽然它似乎是数值型的），但实际上它是一个因子。它根据学生完成多少作业把每个期中成绩分配给五组中的一组。

考试成绩的分布不是正态的：学生数学技能的差异很大，因此成绩为A和F的学生人数很多。因此，常规方差分析是不适合的。于是我使用Kruskal-Wallis检验，并获得了近似于零的p值（3.99×10^{-5}或者0.000 036 69）：

```
> kruskal.test(midterm ~ hw)

        Kruskal-Wallis rank sum test

data: midterm by hw
Kruskal-Wallis chi-squared = 25.6813, df = 4, p-value = 3.669e-05
```

显然，完成作业的学生和不完成作业的学生在考试成绩上有显著的差异。但我事实上得出了什么结论呢？首先，我很高兴作业似乎是很有效的。其次，它使我明白在统计推理中，认为“相关关系暗示着因果关系”是一个经典的错误。也许有很强动机的学生在作业和考试中都做得很好，而懒惰的学生都表现不好。在这种情况下，因果关系的因子是动机的程度，而不是我给学生布置作业的明智选择。最后，我可以得出很简单的结论：完成作业的学生很有可能在期中考试中表现得很好，但我不知道为什么。

11.24 运用方差分析比较模型

问题

有两个针对相同数据的模型，而你想知道它们是否会产生不同的结果。

解决方案

函数anova可以比较两个模型并且显示它们是否显著不同：

```
> anova(m1, m2)
```

这里，m1和m2都是由lm返回的模型对象。函数anova的输出包含一个p值。按照惯例，$p<0.05$值表示模型是显著不同的，而$p>0.05$不提供这样的证据。

讨论

在方法11.3中，我们调用函数anova来打印一个回归模型的方差分析表。现在我们使用两个参数的形式来比较两个模型。

在比较两个模型时，函数anova有一个强烈的要求：一个模型必须包含在另一个中。也就是说，较小模型的所有项必须出现在较大的模型中；否则，比较是不可能的。

方差分析执行F检验，它与线性回归的F检验是相似的，不同的是，该检验是比较两个模型，而回归F检验比较运用回归模型和不运用模型。

假设我们通过增加预测变量的方式对y建立三个模型：

```
> m1 <- lm(y ~ u)
> m2 <- lm(y ~ u + v)
> m3 <- lm(y ~ u + v + w)
```

m2真的与m1不同吗？我们可以调用anova来比较它们，结果的p值为0.003 587：

```
> anova(m1, m2)
Analysis of Variance Table

Model 1: y ~ u
Model 2: y ~ u + v
  Res.Df     RSS Df Sum of Sq      F   Pr(>F)
1     28 108.968
2     27  79.138  1     29.83 10.177 0.003587 **
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

较小的p值表示模型是显著不同的。然而，比较m2和m3，我们得到p值为0.057 44：

```
> anova(m2, m3)
Analysis of Variance Table

Model 1: y ~ u + v
Model 2: y ~ u + v + w
  Res.Df    RSS Df Sum of Sq      F  Pr(>F)
1     27 79.138
2     26 68.696  1    10.442 3.9522 0.05744 .
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

p值接近于我们的阈值0.05。严格地说，它没有满足p值小于0.05的要求；然而，它足够接近，你可能会判断模型“有足够的差异”。

这个例子有点做作，所以它并没有显示方差分析的强大功能。当我调用anova，并同时通过添加和删除多个项来实验复杂的模型时，我需要知道新模型是否与原来的不同。换句话说：如果我添加了新项而新的模型基本上不变，那么额外的项带来的复杂性是不值得的。

第12章

有用的方法

简介

本章中介绍的方法既不是晦涩的数值计算，也不是深厚的统计技术。但它们是有用的函数和语法，你可能时不时地会需要使用它们。

12.1 查看你的数据

问题

有许多数据——多到不能一次性展示出来。然而，你需要查看一些数据。

解决方案

调用函数head来查看最初的几个或行数据：

```
> head(x)
```

调用函数tail来查看最后的几个或行数据：

```
> tail(x)
```

讨论

输出一个大的数据集是毫无意义的，因为它们不会全部展示在屏幕上。调用函数head来查看一部分数据：

```
> head(dfrm)
           x           y          z
1  0.7533110  0.57562846 -0.1710760
2  2.0143547  0.83312274  0.3698584
3 -0.3551345  0.57471542  2.0132348
4  2.0281678  0.78945319 -0.5378854
5 -2.2168745  0.01758024  1.8344879
6  0.7583962 -1.78214755  2.2848990
```

调用函数tail来查看最后的几行数据和总行数。这里，我们发现这个数据框有10120行：

```
> tail(dfrm)
               x           y          z
10115 -0.0314354 -0.74988291 -0.2048963
10116 -0.4779001  0.93407510  1.0509977
10117 -1.1314402  1.89308417  1.7207972
10118  0.4891881 -1.20792811 -1.4630227
10119  1.2349013 -0.09615198 -0.9887513
10120 -1.3763834 -2.25309628  0.9296106
```

另请参阅

关于查看变量内容的结构，请参见方法12.15。

12.2 拓宽你的输出

问题

正在输出很宽的数据集。R把输出设置得很短，让它难以阅读。

解决方案

设置选项width以使在输出窗口中反映列的真实数量：

```
> options(width=numcols)
```

讨论

默认情况下，R设置输出格式为80列宽。当你输出较宽的数据集，而R根据其假定的80列限制来输出每行数据时，这是令人沮丧的。价格昂贵、宽屏的显示器大概可以容纳更多的列。

例如，如果显示120个字符宽，然后你通过设置参数width让R使用整个120列：

```
> options(width=120)
```

现在R会以120个字符而非80个字符来包装它的行，让你方便地查看更宽的数据集。

另请参阅

参见方法3.16更多关于设置选项的内容。

12.3 输出赋值结果

问题

把一个值赋值给一个变量，并想要查看它的值。

解决方案

简单地给赋值表达式加上括号：

```
> x <- 1/pi              # Prints nothing
> (x <- 1/pi)            # Prints assigned value
[1] 0.3183099
```

讨论

通常情况下，当看到你输入一个简单的赋值时，R不会输出。然而，当你给赋值表达式加上括号时，它不再是一个简单的赋值，所以R输出它的值。

另请参阅

参见方法2.1更多关于打印的内容。

12.4 对行和列求和

问题

需要对一个矩阵或者数据框的行或者列求和。

解决方案

调用函数rowSums来求行的和：

```
> rowSums(m)
```

调用函数colSums来求列的和：

```
> colSums(m)
```

讨论

这是一个平凡的方法，但它是如此常见，所以值得一提。例如，当产生包含列总和的显示时，我用这个方法。在这个例子中，daily.prod是本周工厂生产值的记录，我们想要根据产品和天数计算总和：

```
> daily.prod
    Widgets Gadgets Thingys
Mon     179     167     182
Tue     153     193     166
Wed     183     190     170
Thu     153     161     171
Fri     154     181     186
> colSums(daily.prod)
Widgets Gadgets Thingys
    822     892     875
> rowSums(daily.prod)
Mon Tue Wed Thu Fri
528 512 543 485 521
```

这些函数返回一个向量。在这个列求和的例子中，我们可以附加和向量至原矩阵，因此可以整洁地输出原始数据与总和：

```
> rbind(daily.prod, Totals=colSums(daily.prod))
       Widgets Gadgets Thingys
Mon        179     167     182
Tue        153     193     166
Wed        183     190     170
Thu        153     161     171
Fri        154     181     186
Totals     822     892     875
```

12.5 按列输出数据

问题

有多个平行的数据向量，而你想把它们输出在列中。

解决方案

调用函数cbind把数据构建成列，然后打印这个结果。

讨论

当有平行向量时，如果分开输出它们，那么看清它们的关系会是很困难的：

```
> print(x)
 [1] -0.3142317 2.4033751 -0.7182640 -1.7606110 -1.1252812 -0.7195406 1.3102240
 [8]  0.4518985 0.1521774  0.6533708
> print(y)
 [1] -0.9473355 -1.0710863 -0.1760594 1.8720502 -0.5103384 1.4388904 1.1531710
 [8] -0.1548398 -0.2599055  0.4713668
```

调用函数cbind把它们构建成列，当打印时，显示数据的结构：

```
> print(cbind(x,y))
               x          y
 [1,] -0.3142317 -0.9473355
 [2,]  2.4033751 -1.0710863
 [3,] -0.7182640 -0.1760594
 [4,] -1.7606110  1.8720502
 [5,] -1.1252812 -0.5103384
 [6,] -0.7195406  1.4388904
 [7,]  1.3102240  1.1531710
 [8,]  0.4518985 -0.1548398
 [9,]  0.1521774 -0.2599055
[10,]  0.6533708  0.4713668
```

你也可以在输出中包含表达式。使用标签给它们添加一个列标题：

```
> print(cbind(x,y,Total=x+y))
               x          y      Total
 [1,] -0.3142317 -0.9473355 -1.2615672
 [2,]  2.4033751 -1.0710863  1.3322888
 [3,] -0.7182640 -0.1760594 -0.8943233
 [4,] -1.7606110  1.8720502  0.1114392
 [5,] -1.1252812 -0.5103384 -1.6356196
 [6,] -0.7195406  1.4388904  0.7193499
 [7,]  1.3102240  1.1531710  2.4633949
 [8,]  0.4518985 -0.1548398  0.2970587
 [9,]  0.1521774 -0.2599055 -0.1077281
[10,]  0.6533708  0.4713668  1.1247376
```

12.6 对数据分级

问题

有一个向量，需要把数据根据间隔分成组。统计学家称其为对数据分级。

解决方案

调用函数cut。你必须定义一个向量，比如说breaks，它给出这个间隔的范围。函数cut根据间隔对数据分组。它返回一个因子，它的水平（元素）识别每个基准的组：

```
> f <- cut(x, breaks)
```

讨论

这个例子产生1000个标准正态分布的随机数。它通过定义间隔为±1、±2和±3个标准偏差，把这些数据分成6个组：

```
> x <- rnorm(1000)
> breaks <- c(-3,-2,-1,0,1,2,3)
> f <- cut(x, breaks)
```

结果是一个因子f，它确定了分组。函数summary根据水平显示该组元素的数量。R为每个水平创建名称，对间隔使用数学符号：

```
> summary(f)
(-3,-2] (-2,-1]  (-1,0]   (0,1]   (1,2]   (2,3]    NA's
     18     119     348     355     137      21       2
```

结果是我们所期望的钟形。它有两个NA值，这表明x中的两个值落在定义的区间外。

我们可以使用labels参数把预定义的名称分配给这6个组而不是将晦涩的合成的名称分配给它们：

```
> f <- cut(x, breaks, labels=c("Bottom", "Low", "Neg", "Pos", "High", "Top"))
```

现在，函数summary使用我们定义的名称：

```
Bottom    Low    Neg    Pos   High    Top   NA's
    18    119    348    355    137     21      2
```

> **不要扔掉信息**
>
> 分组对总结是很有用的，例如对于直方图。但它会导致信息丢失，这在建模中是有害的。考虑到在极端情况下，将一个连续变量分级为两个值，“高”值和“低”值。分级数据只有两种可能值，因此少用了一些信息源。其中，连续变量可能是一个强大的预测变量，分级变量可以区分最多两个变量，因此有可能只有原来的少部分信息。在分级之前，我建议探索其他信息损耗少的转换。

12.7 找到特定值的位置

问题

有一个向量。你知道在内容中有一个特定的值，而你想知道它的位置。

解决方案

函数match在向量中搜索一个特定值并返回它的值：

```
> vec <- c(100,90,80,70,60,50,40,30,20,10)
> match(80, vec)
[1] 3
```

这里，match返回3，它是vec中80这个值的位置。

讨论

对于找出最小值和最大值的位置，R中有特殊的函数——它们分别是which.min和which.max：

```
> vec <- c(100,90,80,70,60,50,40,30,20,10)
> which.min(vec)            # Position of smallest element
[1] 10
> which.max(vec)            # Position of largest element
[1] 1
```

另请参阅

这个方法也运用在方法11.12中。

12.8 每隔*n*个选定一个向量元素

问题

需要每隔*n*个值选定向量中的一个元素。

解决方案

创建一个逻辑索引向量，每隔*n*个元素将它的值设置为TRUE。一种方法是当进行模*n*运算时，找到所有等于零的下标：

```
> v[ seq_along(v) %% n == 0 ]
```

讨论

这个问题出现在系统抽样中：我们要通过选定每隔*n*个元素的值来对一个数据集抽样。函数seq_along(v)生成可以索引v的整数数列，它相当于1:length(v)。我们通过以下表达式计算每个索引值的模*n*运算：

```
seq_along(v) %% n
```

然后我们找到值等于0的索引值：

```
seq_along(v) %% n == 0
```

该结果是一个逻辑向量，它和v有相同长度并且对每隔*n*个元素为TRUE，它可以用来索引v以选定需要的元素：

```
v[ seq_along(v) %% n == 0 ]
```

如果你只是简单地想要每隔两个元素的值，你可以调用循环规则。用一个二元素逻辑向量来索引v，如下所示。

```
v[ c(FALSE, TRUE) ]
```

如果v中有多于两个元素，那么索引向量太短了。因此R会调用循环规则并且把索引向量的长度扩充到v的长度，循环它的内容。这给出了一个索引向量，它是FALSE、TRUE、FALSE、TRUE、FALSE、TRUE等。所以，最后的结果是v的每隔两个元素的值。

另请参阅

参见方法5.3更多关于循环规则的内容。

12.9 找到成对的最小值或者最大值

问题

有两个向量v和w，而你想要找到成对元素的最小值或者最大值。即计算：

$$\min(v_1, w_1), \min(v_2, w_2), \min(v_3, w_3), \ldots$$

或者：

$$\max(v_1, w_1), \max(v_2, w_2), \max(v_3, w_3), \ldots$$

解决方案

R称这些为平行最小值和平行最大值。分别由函数pmin(v,w)和pmax(v,w)计算：

```
> pmin(1:5, 5:1)        # Find the element-by-element minimum
[1] 1 2 3 2 1
> pmax(1:5, 5:1)        # Find the element-by-element maximum
[1] 5 4 3 4 5
```

讨论

当R初学者想要成对的最小值或最大值时，一个常见的错误是：写出min(v,w)或者max(v,w)。这些都不是成对的操作：min(v,w)返回单个值，所有v和w的最小值。同样，max(v,w)从所有v和w中返回单个值。

值pmin和pmax平行比较它们的参数，对每个下标计算出最小值或者最大值。它们返回一个向量，它匹配输入的长度。

你可以用循环规则结合pmin和pmax来执行这个有用的方法。假设向量v包含正值和负值，而你要重置负值为零。方法如下：

```
> v <- pmax(v, 0)
```

通过循环规则，R扩展这个零值纯量为一个长度与v相同的零向量，然后pmax逐个元素比较它们，取出v的每个元素和零值中较大的值：

```
> v <- c(1, -2, 3, -4, 5, -6, 7)
> pmax(v,0)
[1] 1 0 3 0 5 0 7
```

事实上，pmin和pmax的功能比解决方案所描述的更加强大。它们可以处理多于两个的向量，平行比较所有的向量。

另请参阅

更多关于循环规则的内容，参见方法5.3。

12.10 生成多个因子的组合

问题

有两个或者更多个因子。你想要生成这些水平的所有组合，也称为它们的笛卡儿积。

解决方案

调用函数expand.grid。这里f和g是两个因子：

```
> expand.grid(f, g)
```

讨论

下面的代码段创建了两个因子——因子sides代表一个硬币的两面，而因子faces代表一个骰子的六面（骰子面上的小斑点称为点）：

```
> sides <- factor(c("Heads", "Tails"))
> faces <- factor(c("1 pip", paste(2:6, "pips")))
```

可以使用expand.grid来找到掷一次骰子和掷一次硬币的结果的所有组合：

```
> expand.grid(faces, sides)
     Var1  Var2
1   1 pip Heads
2  2 pips Heads
3  3 pips Heads
4  4 pips Heads
5  5 pips Heads
6  6 pips Heads
7   1 pip Tails
8  2 pips Tails
9  3 pips Tails
10 4 pips Tails
11 5 pips Tails
12 6 pips Tails
```

同样，我们可以找到掷两个骰子的结果的所有组合：

```
> expand.grid(faces, faces)
```

```
     Var1   Var2
1   1 pip 1 pip
2 2 pips 1 pip
3 3 pips 1 pip
4 4 pips 1 pip
5 5 pips 1 pip
6 6 pips 1 pip
7   1 pip 2 pips
8 2 pips 2 pips
.
. (etc.)
.
34 4 pips 6 pips
35 5 pips 6 pips
36 6 pips 6 pips
```

结果存于一个数据框中。R自动提供行名和列名。

这里的解决方案和例子给出的是两个因子的笛卡儿积，但expand.grid也可以处理三个或者更多个的因子。

另请参阅

如果你正在处理字符串而不是因子，那么你也可以调用方法7.7来生成组合。

12.11 转换一个数据框

问题

有一个数值型数据框。你想要一起处理它所有的元素，而不是分成列处理，例如，找到所有值的均值。

解决方案

把数据框转换为一个矩阵并处理这个矩阵。该例子找到在数据框dfrm中所有元素的均值：

```
> mean(as.matrix(dfrm))
```

某些情况下有必要把矩阵转换为一个向量。在这种情况下，调用函数as.vector(as.matrix(dfrm))。

讨论

假设我们有一个数据框，例如方法12.4中的工厂产量数据：

```
> daily.prod
    Widgets Gadgets Thingys
Mon     179     167     182
Tue     153     193     166
Wed     183     190     170
Thu     153     161     171
Fri     154     181     186
```

假设我们需要所有天以及产品的平均每日产量，以下代码不会奏效：

```
> mean(daily.prod)
Widgets Gadgets Thingys
164.4   178.4   175.
```

函数mean认为应该对每列取平均，这通常是你想要的结果。但当你需要所有值的均值时，应该先把数据框转换成一个矩阵：

```
> mean(as.matrix(daily.prod))
[1] 172.6
```

这个方法只能处理包含数值型数据的数据框。前面讲过转换一个有混合数据类型的数据框（数值型列、字符型列或者因子的混合）为一个矩阵时，需要强制转换每列为字符型。

另请参阅

更多关于在两个数据类型之间转换的内容，请参见方法5.33。

12.12 对数据框排序

问题

有一个数据框。你需要通过使用一列作为排序关键字将它的内容排序。

解决方案

调用函数order以得到排序关键字，然后根据它的顺序重新安排数据框的行：

```
> dfrm <- dfrm[order(dfrm$key),]
```

这里，dfrm是一个数据框，而dfrm$key是排序关键字列。

讨论

函数sort适用于处理向量，而对数据框无效。对一个数据框排序需要一个额外的步骤。假设我们有以下数据框，而我们需要根据月份对它们排序：

```
> print(dfrm)
  month day outcome
1     7  11     Win
2     8  10    Lose
3     8  25     Tie
4     6  27     Tie
5     7  22     Win
```

函数order告诉我们如何对月份按照升序重新排列：

```
> order(dfrm$month)
[1] 4 1 5 2 3
```

它返回的不是数据值，而是dfrm$month实际排序中相应于原始数据的下标。我们使用这些下标重新排序整个数据框：

```
> dfrm[order(dfrm$month),]
  month day outcome
4     6  27     Tie
1     7  11     Win
5     7  22     Win
2     8  10    Lose
3     8  25     Tie
```

在重新排序数据框之后，月份列为我们希望的升序排列。

12.13 对两列排序

问题

需要对一个数据框的内容排序，它使用两列数据作为排序关键字。

解决方案

根据两列排序与根据一列排序的方法相似。函数order接受多个参数，所以我们可以给它两个排序关键字：

```
> dfrm <- dfrm[order(dfrm$key1,dfrm$key2),]
```

讨论

在方法12.12中，使用函数order对数据框的列重新排序。我们可以把第二个排序关键字传给函数order，函数order使用这个额外的排序关键字打破第一个排序关键字中存在的并列关系。

继续方法12.12的例子，我们可以用order对month和day这两列进行排序：

```
> dfrm[order(dfrm$month,dfrm$day),]
  month day outcome
4     6  27     Tie
1     7  11     Win
5     7  22     Win
2     8  10    Lose
3     8  25     Tie
```

在7月份和8月份中，现在按照升序对天数进行排列。

另请参阅

参见方法12.12。

12.14 移除变量属性

问题

一个变量含有一些旧属性。你需要移除它们中的一些或者全部。

解决方案

对变量的attributes属性指定NULL值，可以移除所有属性：

```
> attributes(x) <- NULL
```

调用函数attr来选定单个属性，把它设置为NULL，以移除单个属性：

```
> attr(x, "attributeName") <- NULL
```

讨论

R中的任何变量都可以有属性。属性是一个简单的名称和值对，变量可以有很多属性。一个常见的例子是一个矩阵变量的维，它存储在一个属性中。属性名是dim而属性值是一个二元素向量，它给出了行和列的数目。

你可以通过输出attributes(x)或者str(x)来查看x的属性。

有时你只想要一个数字，R坚持给出它的属性。当拟合一个简单的线性模型，并且提取斜率时，这可能会发生，它是第二个回归系数：

```
> m <- lm(resp ~ pred)
> slope <- coef(m)[2]
> slope
    pred
1.011960
```

当我们输出slope时，R同时输出"pred"。它是由lm给出的回归系数的名称属性（因为它是变量pred的系数）。我们可以通过输出slope的内部内容来更清晰地看到，它显示一个"names"属性：

```
> str(slope)
Named num 1.01
- attr(*, "names")= chr "pred"
```

移除所有属性是简单的，移除属性之后，斜率的值会变成一个数字：

```
> attributes(slope) <- NULL    # Strip off all attributes
> str(slope)                   # Now the "names" attribute is gone
num 1.01
> slope                        # And the number prints cleanly without a label
[1] 1.011960
```

另外，我们也可以用以下代码移除单个属性：

```
> attr(slope, "names") <- NULL
```

警告： 请记住，矩阵是一个含有dim属性的向量（或列表）。如果你移除一个矩阵的所有属性，会移除它的维数，从而把它变成一个纯粹的向量（或列表）。此外，移除一个对象的属性（尤其是S3对象），该对象可能会失去作用。所以小心删除属性。

另请参阅

更多关于查看属性的内容，参见方法12.15。

12.15 显示对象的结构

问题

调用一个函数，然后返回结果。现在你需要查看这个结果并且了解它更多的内容。

解决方案

调用class获取对象类：

```
> class(x)
```

调用mode移除面向对象的特性，并显示潜在的结构：

```
> mode(x)
```

调用str显示内在结构和内容：

```
> str(x)
```

讨论

很多时候，我调用一个函数，得到返回的结果，然后疑惑："这到底是什么呢？"从理论上讲，函数的文档应该解释返回值，但不知何故，当我自己查看它的结构和内容时，感觉会更好。对于有着嵌套结构的对象（对象内含有对象），这尤其正确。

让我们来分析lm（线性建模函数）在最简单的线性回归方法中的返回值，参见方法11.1：

```
> m <- lm(y ~ x)
> print(m)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
      17.72         3.25
```

我总是通过检查对象的类来开始处理数据。类表示它是否是一个向量、矩阵、列表、数据框或者对象：

```
> class(m)
[1] "lm"
```

似乎m是一个lm类的对象。这对我并不意味着什么。然而，我知道所有的对象类根据内在的数据结构（向量、矩阵、列表或数据框）来构建，所以我调用mode来移除对象的外观，显示它的内在基本结构：

```
> mode(m)
[1] "list"
```

看来m是列表结构。现在我可以使用列表的函数和运算符来查看它的内容。首先，我想知道这个列表的元素的名称：

```
> names(m)
 [1] "coefficients" "residuals"    "effects"      "rank"         "fitted.values"
 [6] "assign"       "qr"           "df.residual"  "xlevels"      "call"
[11] "terms"        "model"
```

列表的第一个元素称为“系数”。我猜想它们是回归系数，如下所示。

```
> m$coefficients
(Intercept)           x
17.722623    3.249677
```

的确，它们是回归系数。我识别出了这些值。

我可以继续查看m的列表结构，但这会很冗长。函数str适用于处理显示任何变量的内在结构。m的输出结果如例子12-1所示。

例12-1：返回一个变量的结构

```
> str(m)
List of 12
 $ coefficients : Named num [1:2] 17.72 3.25
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "x"
 $ residuals    : Named num [1:30] -12.471 -3.983 -2.899 0.192 14.302 ...
  ..- attr(*, "names")= chr [1:30] "1" "2" "3" "4" ...
 $ effects      : Named num [1:30] -372.443 155.463 -0.614 2.418 16.552 ...
  ..- attr(*, "names")= chr [1:30] "(Intercept)" "x" "" "" ...
 $ rank         : int 2
 $ fitted.values: Named num [1:30] 17.9 23.9 26.8 32.2 30 ...
  ..- attr(*, "names")= chr [1:30] "1" "2" "3" "4" ...
 $ assign       : int [1:2] 0 1
 $ qr           :List of 5
  ..$ qr   : num [1:30, 1:2] -5.477 0.183 0.183 0.183 0.183 ...
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:30] "1" "2" "3" "4" ...
  .. .. ..$ : chr [1:2] "(Intercept)" "x"
  .. ..- attr(*, "assign")= int [1:2] 0 1
  ..$ qraux: num [1:2] 1.18 1.23
  ..$ pivot: int [1:2] 1 2
  ..$ tol  : num 1e-07
  ..$ rank : int 2
  ..- attr(*, "class")= chr "qr"
 $ df.residual  : int 28
 $ xlevels      : list()
```

```
 $ call        : language lm(formula = y ~ x)
 $ terms       :Classes 'terms', 'formula' length 3 y ~ x
  .. ..- attr(*, "variables")= language list(y, x)
  .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. ..$ : chr [1:2] "y" "x"
  .. .. .. ..$ : chr "x"
  .. ..- attr(*, "term.labels")= chr "x"
  .. ..- attr(*, "order")= int 1
  .. ..- attr(*, "intercept")= int 1
  .. ..- attr(*, "response")= int 1
  .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. ..- attr(*, "predvars")= language list(y, x)
  .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
  .. .. ..- attr(*, "names")= chr [1:2] "y" "x"
 $ model :'data.frame': 30 obs. of 2 variables:
  ..$ y: num [1:30] 5.41 19.94 23.92 32.43 44.26 ...
  ..$ x: num [1:30] 0.0478 1.9086 2.7999 4.4676 3.7649 ...
  ..- attr(*, "terms")=Classes 'terms', 'formula' length 3 y ~ x
  .. .. ..- attr(*, "variables")= language list(y, x)
  .. .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:2] "y" "x"
  .. .. .. .. ..$ : chr "x"
  .. .. ..- attr(*, "term.labels")= chr "x"
  .. .. ..- attr(*, "order")= int 1
  .. .. ..- attr(*, "intercept")= int 1
  .. .. ..- attr(*, "response")= int 1
  .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. .. ..- attr(*, "predvars")= language list(y, x)
  .. .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
  .. .. .. ..- attr(*, "names")= chr [1:2] "y" "x"
 - attr(*, "class")= chr "lm"
```

注意，str显示m的所有元素，然后递归地给出每个元素的内容和属性。长向量和列表被截断以使输出便于管理。

探索R对象是有艺术的。使用class、mode和str查看R对象的层次。

12.16 代码运行时间

问题

你想知道需要多少时间来运行你的代码。例如，当你优化你的代码，并且需要得到优化前和优化后的时间，据此衡量优化是否有效时，这是非常有用的。

解决方案

函数system.time执行代码并且返回执行时间：

```
> system.time(aLongRunningExpression)
```

它输出3~5个数字，这根据你的平台而定。前3个数字往往是最重要的：

用户CPU时间

运行R的CPU秒数

系统CPU时间

运行操作系统（通常为输入/输出）的CPU秒数

经过的时间

时钟秒数

讨论

假设我们想知道生成10 000 000个随机正态变量并对它们求和所需要的时间：

```
> system.time(sum(rnorm(10000000)))
[1] 3.06 0.01 3.14    NA    NA
```

system.time的输出显示R使用的CPU时间为3.06秒；操作系统使用的CPU时间为0.01秒；整个测试过程经过了3.14秒。（NA值是子进程时间；在这个例子中，没有子进程时间。）

第三个值，经过时间，不是用户CPU时间和系统CPU时间的简单总和。如果计算机忙于运行其他进程，那么会使用有更多时间，因为R与其他进程共享CPU。

12.17 抑制警告和错误消息

问题

函数产生恼人的警告和错误消息。你不想看见它们。

解决方案

对该函数调用suppressMessage(...)或suppressWarnings(...)：

```
> suppressMessage(annoyingFunction())
> suppressWarnings(annoyingFunction())
```

讨论

我经常调用函数adf.test。然而，当p值小于0.01时，它产生一个恼人的警告信息，例如下面输出的底部显示：

```
> library(tseries)
> adf.test(x)

        Augmented Dickey-Fuller Test

data:   x
Dickey-Fuller = -4.3188, Lag order = 4, p-value = 0.01
alternative hypothesis: stationary

Warning message:
In adf.test(x) : p-value smaller than printed p-value
```

幸运的是，我可以通过在suppressWarnings(...)里调用它来包装这个函数：

```
> suppressWarnings(adf.test(x))

        Augmented Dickey-Fuller Test

data:  x
Dickey-Fuller = -4.3188, Lag order = 4, p-value = 0.01
alternative hypothesis: stationary
```

注意，警告信息消失了。这个消息并非完全消失，因为R在内部保留它。可以通过调用函数warnings在闲暇时检索这个消息：

```
> warnings()
Warning message:
In adf.test(x) : p-value smaller than printed p-value
```

其他函数也会产生“消息”（R术语），比起警告它更多是良性的。通常，它们只是信息，而不是问题信号。如果这样的消息让你烦恼，在suppressMessage()内调用该函数，然后消息将会消失。

另请参阅

参见函数options关于其他控制错误和警告信息报告的方法。

12.18 从列表中提取函数参数

问题

你的数据显示在列表结构中。你需要把数据传递给一个函数，但该函数不接受列表。

解决方案

在简单情况下，把列表转换为一个向量。对于更加复杂的情况，函数do.call会把列表拆分为单个参数并且调用你的函数：

```
> do.call(function, list)
```

讨论

如果数据用向量表示，处理它就很简单，大多数情况下，R会像你期望的那样运行：

```
> vec <- c(1, 3, 5, 7, 9)
> mean(vec)
[1] 5
```

如果数据是用列表来表示，那么某些函数会失效，它们将返回一个无用的结果，如下所示。

```
> numbers <- list(1, 3, 5, 7, 9)
> mean(numbers)
[1] NA
Warning message:
In mean.default(numbers) : argument is not numeric or logical: returning NA
```

列表numbers是一个简单的，单个水平的列表，所以我们可以把它转换为一个向量并且调用这个函数：

```
> mean(unlist(numbers))
[1] 5
```

当你有多个水平的列表结构（列表中有列表）时，情况就变得复杂了。这种情况会在复杂数据结构中出现。这里，包含多个列表的列表中的每个子表都是一个数据列：

```
> lists <- list(col1=list(7,8,9), col2=list(70,80,90), col3=list(700,800,900))
```

假设我们需要把这个数据转换成一个矩阵。函数cbind应该创建数据列，但它不明白列表结构并且返回了无用的结果：

```
> cbind(lists)
     lists
col1 List,3
col2 List,3
col3 List,3
```

如果我们用unlist来处理数据，那么我们能得到一个大而长的列：

```
> cbind(unlist(lists))
      [,1]
col11    7
col12    8
col13    9
col21   70
col22   80
col23   90
col31  700
col32  800
col33  900
```

解决方案是调用do.call，它把列表分拆为单个项，然后对这些单项调用cbind：

```
> do.call(cbind, lists)
     col1 col2 col3
[1,] 7    70   700
[2,] 8    80   800
[3,] 9    90   900
```

从功能上来看，用这个方法调用do.call与下面调用cbind是一样的：

```
> cbind(lists[[1]], lists[[2]], lists[[3]])
     [,1] [,2] [,3]
[1,] 7    70   700
[2,] 8    80   800
[3,] 9    90   900
```

如果列表元素有名称，那就要小心处理。在这种情况下，do.call把元素名称解释为函数的参数名称，这会导致麻烦。

本方法给出了do.call最基本的用途。函数do.call的功能十分强大并且有许多其他用途。详情参见帮助说明页。

另请参阅

更多关于在数据类型间转换的内容,请参见方法5.33。

12.19 定义你自己的二元运算符

问题

你要定义你自己的二元运算符，让R代码更流畅、更易读。

解决方案

R认为任何在百分比符号(%...%)间的文本为二元运算符。通过对它赋值一个两参数函数，就可以创建并定义一个新的二元运算符。

讨论

R包含一个有趣的功能，可以让你定义你自己的二元运算符。任何在百分比符号(%...%)间的文本将自动由R解释为二元运算符。R预定义几个这样的运算符，如对于整数除法的%%和对于矩阵乘法的%*%。

通过指定函数，你可以创建一个新的二元运算符。以下例子创建一个算符，%+－%：

```
> '%+-%' <- function(x,margin) x + c(-1,+1)*margin
```

表达式x %+－% m计算x±m。下面的代码计算100±(1.96 × 15)，标准IQ检验的两个标准偏差范围：

```
> 100 %+-% (1.96*15)
[1] 70.6 129.4
```

注意，当我们在定义二元运算符时，需要用引用符。而使用它时则不需要引用符。

定义自己的运算符的乐趣在于，你通常可以把运算符包装在一个简洁的语法中。如果你的应用程序经常没有中间空格地连接字符串，那么你可能为此目的定义一个二元连接运算符：

```
> '%+%' <- function(s1,s2) paste(s1,s2,sep="")
> "Hello" %+% "World"
[1] "HelloWorld"
> "limit=" %+% round(qnorm(1-0.05/2), 2)
[1] "limit=1.96"
```

定义自己的运算符的危险在于，代码不能移植到其他环境。需要把定义运算符的代码一起放到应用运算符的地方；否则，R将对使用未定义的运算符产生错误信息。

所有用户定义的运算符有相同的优先级并集体列于表2-1中（其中用%*any*%表示用户定义运算符）。它们的优先级是相当高的：高于乘法和除法，但低于幂和数列创建。这样高优先级的一个结果是，很容易造成误解。如果我们在上面应用"%+－%"的例子中省略了括号，我们会得到意想不到的结果：

```
> 100 %+-% 1.96*15
[1] 1470.6 1529.4
```

R把该表达式解释为`(100 %+-% 1.96) * 15`。

另请参阅

更多关于运算符优先级的内容，请参见方法2.11；关于如何定义一个函数的内容，请参见方法2.12。

第13章

高级数值分析和统计方法

简介

本章将给出几个应用统计中的高级方法，这些方法在研究生课程中将会学习到。

这些方法大部分在R的基础发布版中。通过添加包的形式，R提供了当今最先进的统计技术。因为R现在成为统计学家事实上的必用工具，所以他们用R来实现他们的最新研究工作。任何人如果需要寻找最先进的统计技术，这里极力推荐首先到R的官网CRAN或者其他R相关的网站上寻找。

13.1 最小化或者最大化一个单参数函数

问题

给定一个单参数函数f(x)，需要找到一个点x使函数取值为最小或者最大。

解决方案

函数optimize可以最小化一个单参数函数。它指明需要最小化的函数f和该函数的定义域（即x的上界和下界）。

```
> optimize(f, lower=lowerBound, upper=upperBound)
```

如果需要最大化函数f，需要指明optimize的参数maximum=TRUE：

```
> optimize(f, lower=lowerBound, upper=upperBound, maximum=TRUE)
```

讨论

函数optimize能够对单参数的函数求极大值或者极小值。它需要在参数中指明需要求极值的函数的自变量x的取值范围，这将决定极大值点或者极小值点的搜索范围。下面的例子是求多项式函数$3x^4 - 2x^3 + 3x^2 - 4x + 5$的极小值：

```
> f <- function(x) 3*x^4 - 2*x^3 + 3*x^2 - 4*x + 5
> optimize(f, lower=-20, upper=20)
$minimum
[1] 0.5972778

$objective
[1] 3.636756
```

函数optimize的返回值是一个含有两个元素的列表：其中minimum代表使得函数取到最小值的x的值；objective为函数在该点所达到的极小值。

如果参数lower和upper的距离较小，它意味着搜索的范围较小，最优化的速度将较快。如果你不确定什么是合适的搜索范围，建议采用一个合理的、较大的范围，例如取lower=－1000和upper=1000。需要注意的是，函数在该区间不要有多个最小值或者最大值，函数optimize只返回一个最小值或者最大值。

另请参阅

参见方法13.2。

13.2 最小化或者最大化多参数函数

问题

对于多参数（有多个自变量）的函数f，需要找到使该函数取得最小值或者最大值的自变量的取值。

解决方案

用函数optim来对多参数函数f进行极小化。这里需要设定optim的参数t，它给定一个搜索的起始点，t的取值为函数f的参数的初始值。

```
> optim(startingPoint, f)
```

如果需要最大化函数，需要设定参数control。

```
> optim(startingPoint, f, control=list(fnscale=-1))
```

讨论

由于函数optim可以用于多个参数的函数，所以函数optim是比optimize更通用的一个函数（参见方法13.1）。函数optim把函数f的自变量的取值放到一个向量中，然后估计函数在该向量上的取值。函数的取值是一个纯量值。函数optim将从设定的初始点开始，在自变量的定义域内搜索函数的最小值。

下面给出一个用optim来拟合非线性模型的例子。假设你认为变量对z和x通过函数$z_i=(x_i+a)+\beta+\varepsilon_i$相关联，其中$\alpha$和$\beta$是两个未知的参数，$\varepsilon_i$是非正态分布的噪声项。下面通过最小化一个稳健的指标，即绝对偏离和，来拟合该模型：

$$\Sigma|z-(x+\alpha)^b|$$

下面首先定义需要最小化的函数f，它有一个形式上的参数v，该参数是一个二元素的向量。实际需要估计的参数是a和b，这里把它们分别放到向量v的第一个和第二个元素的位置。

```
f <- function(v) { a <- v[1]; b <- v[2]          # "Unpack" v, giving a and b
                   sum(abs(z - ((x + a)^b)))
                 }
```

调用函数optim从点(1, 1)开始进行搜索，寻找f的极小值：

```
> optim(c(1,1), f)
$par
[1] 10.0072591 0.6997801

$value
[1] 1.26311

$counts
function gradient
     485       NA

$convergence
[1] 0

$message
NULL
```

函数optim的返回值包含一个convergence，它的取值将说明函数optim是否找到了f的最小值。如果取值为0，则说明optim找到了最小值；否则，说明没有找到最小值。毫无疑问，convergence的值是最重要的返回值，如果convergence的值为1，说明函数optim不收敛（即没有找到最小值），那么其他的返回值是没有任何意义的。

在本例中，函数optim是收敛的并且找到了一个最小值，大约在$a = 10.01$和 $b = 0.700$函数取到最小值。

对于函数optim而言，它没有参数指明搜索的上边界和下边界，仅仅需要给定一个搜索的起始点即可。对起始点的一个好的猜测意味着最优化算法的加快。

函数optim支持多种不同的最优化算法，你可以在其中选择一个。如果默认的算法无效，查看函数的帮助文档获取其他算法的信息。多变量最优化算法经常遇到的问题是它们经常仅仅给出一个局部最小值(或者最大值)点，而不是找到更深层次的全局的最小值（或最大值）点。一般来说，越是强大有效的算法，越容易给出全局最优点。当然，这里有个有效性和速度的折中，强大有效的算法能找到全局最优点，但是收敛速度较慢。

另请参阅

R社区实现了许多最优化的工具。请查阅R网站CRAN的任务概览部分的“Optimization and Mathematical Programming”部分，以获得更多解决方案（*http://cran.r-priject.org/web/views/Optimization.html*）。

13.3 计算特征值和特征向量

问题

需要计算矩阵的特征值和特征向量。

解决方案

应用函数eigen，它返回一个含有两个元素的列表，这两个元素分别为values和vectors，它们分别表示矩阵的特征值和特征向量。

讨论

假设有一个矩阵，例如斐波那契矩阵。

```
> fibmat
     [,1] [,2]
[1,]    0    1
[2,]    1    1
```

对于给定的矩阵，函数eigen返回含有该矩阵的特征值和特征向量的列表:

```
> eigen(fibmat)
$values
[1]  1.618034 -0.618034
```

```
$vectors
          [,1]       [,2]
[1,] 0.5257311 -0.8506508
[2,] 0.8506508  0.5257311
```

用eigen(fibmat)$values或者eigen(fibmat)$vectors从列表中选择所需要的列表元素。

13.4 主成分分析

问题

现有一个多变量的数据集，需要找出变量的主成分。

解决方案

应用函数prcomp，该函数的第一个参数是一个公式，公式的右边是用“+”号分割的变量集，公式的左边是空白，例如：

```
> r <- prcomp( ~ x + y + z)
> summary(r)
```

讨论

R的基础发布版有两个用于主成分分析的函数：函数prcomp和princomp。帮助文档提到函数prcomp有较好的数值属性，因此这里选择该函数。

应用主成分分析（Principal Components Analysis, PCA）的目的是减少数据集的维数。假设数据集含有N个变量，理想的情况是它们之间是基本独立的，并且贡献了大致相等比例的信息。但是，如果你怀疑某些变量是冗余的，那么主成分分析可以给出数据变差来源的变量个数，如果这个个数接近N，说明所有的变量都是有用的。如果它小于N，那么数据集可以减小为一个较小维数的数据集。

主成分分析把原始的数据集转换到另外一个向量空间，该向量空间的第一个维度将获取最大的变差（即方差最大），第二个维度获取第二大的变差等。主成分分析的输出结果是一个对象，输出时将给出需要的向量旋转。

```
> r <- prcomp( ~ x + y)
> print(r)
Standard deviations:
[1] 0.3724471 0.1306414

Rotation:
         PC1        PC2
x -0.5312726 -0.8472010
y -0.8472010  0.5312726
```

主成分分析的汇总信息十分有用，它会给出每个主成分所获取的变差（或方差）比例。

```
> summary(r)
Importance of components:
                         PC1   PC2
Standard deviation     0.372 0.131
Proportion of Variance 0.890 0.110
Cumulative Proportion  0.890 1.000
```

在这个例子中，第一个主成分获取了89%的变差，第二个主成分获取了11%的变差。因此，第一个主成分获取了大部分的变差。

调用函数prcomp后，应用函数plot(r)来查看主成的方差的直方图；调用函数predict(r)将数据旋转到主成分上。

另请参阅

另外一个应用主成分分析的例子，参见方法13.9。参阅Venables和Ripley《Modern Applied Staticstic with S-Plus》（Springer）一书中关于R的主成分分析的更多内容。

13.5 简单正交回归

问题

应用正交回归来建立线性模型，其中变量x和y的作用是对称的。

解决方案

应用函数prcomp对变量x和y进行主成分分析。从结果的主成分载荷中计算回归的斜率和截距。

```
> r <- prcomp( ~ x + y )
> slope <- r$rotation[2,1] / r$rotation[1,1]
> intercept <- r$center[2] - slope*r$center[1]
```

讨论

正交回归又称为总体最小二乘（Total Least Squares，TLS）。

普通最小二乘（Ordinary Least Squares，OLS）有一个奇怪的性质：它是非对称的，即计算lm(y~x) 在数学上并不是lm(x~y)的逆运算。其原因在于普通最小二乘假设变量x的值是常数，而变量y的值是随机变量，因而所有的变差（即方差）都归于变量y，变量x与变差无关。因此，它是非对称的。

图13-1给出了这种非对称性的示例图。图13-1a是拟合`lm(y~x)`的示意图，普通最小二乘法是最小化图中点和实直线之间的距离，即图中所示的虚线。图13-1b是应用同一个数据集来拟合`lm(x~y)`，因此该模型将最小化水平虚线的距离。很明显，对于最小化的距离不同，得到的结果也不同。

图13-1c不同于图13-1a和b，它是用主成分分析来进行正交回归。这里最小化的距离是点和拟合回归直线间的正交距离。这里是对称的，互换*x*和*y*的角色不会改变最小化的距离。

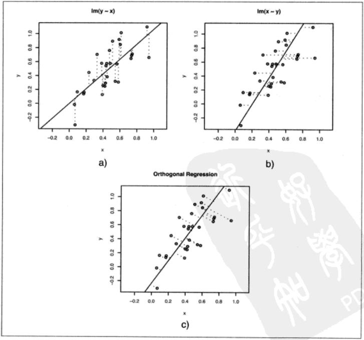


图13-1：普通最小二乘和正交回归

基本的正交回归可以在R中十分简单地实现。首先，进行主成分分析：

```
> r <- prcomp( ~ x + y)
```

用主成分载荷计算斜率：

```
> slope <- r$rotation[2,1] / r$rotation[1,1]
```

然后从斜率来计算截距：

```
> intercept <- r$center[2] - slope*r$center[1]
```

这里称它为“基本”的正交回归，原因在于它仅仅给出了斜率和截距的点估计，而没有给出相应的置信区间。当然，我们还需要回归相关的统计量。方法13.8给出了应用自助法来估计置信区间的方法。

另请参阅

方法13.4讨论了主成分分析。图13-1受到了Vincent Zoonekynd的工作和他的回归分析指南文档的启发（*http://zoonek2.free.fr/UNIX/48_R/09.html*）。

13.6 数据的聚类

问题

你相信数据包含聚类（或者组别）：组内的点之间很“接近”。现在需要找到这些聚类（或者组别）。

解决方案

假设数据集为x，它可以是一个向量、数据框，或者矩阵。另外，假设需要的聚类个数是n。

```
> d <- dist(x)                  # Compute distances between observations
> hc <- hclust(d)               # Form hierarchical clusters
> clust <- cutree(hc, k=n)      # Organize them into the n largest clusters
```

结果clust是一个元素取值为1～n的整数向量，每个整数值对应x中的一个观测值，它给出该观测值所属的类别。

讨论

函数dist 计算所有观测值之间的距离。默认的距离度量为欧几里德距离，该距离适用于大部分的应用。除了欧几里德距离外，还可以选择其他度量距离的方式。

函数hclust根据计算的距离把观测值形成一个层次聚类树。你可以绘制hclust的输出结果来可视化这个层次聚类树，这个树又称为谱系树。如图13-2所示。

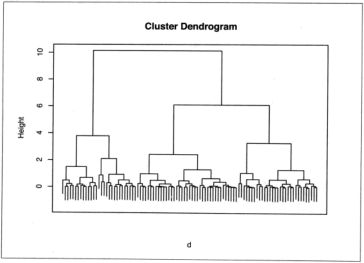


图13-2：绘制谱系树

最后，函数cutree会从谱系树中抽取出聚类。你需要指出你需要多少聚类，或者指出要在什么高度来截断谱系树。一般情况下，聚类的个数是未知的，我们需要仔细检查要聚类的数据来决定对应用有意义的类的个数。

这里我们用一个虚拟合成的数据集来说明数据聚类。首先生成99个服从正态分布的变量值，每一个变量是服从均值为-3、0，或者3的正态分布。

```
> means <- sample(c(-3,0,+3), 99, replace=TRUE)
> x <- rnorm(99, mean=means)
```

出于好奇，你可以计算原始聚类的真实均值（在实际的应用中，我们不知道实际聚类的均值，因此无法计算）。可以验证，这几个聚类的均值相当接近于-3、0和+3。

```
> tapply(x, factor(means), mean)
        -3          0          3
-3.0151424 -0.2238356  2.7602507
```

为了找到聚类，首先计算所有点之间的距离：

```
> d <- dist(x)
```

然后创建层次聚类：

```
> hc <- hclust(d)
```

现在可以抽取三个最大的聚类。显然，我们已经知道真实聚类的个数，这里就有了一个巨大的优势。下面就变得很简单了：

```
> clust <- cutree(hc, k=3)
```

返回值clust的元素是在1～3的整数值，每一个整数值对应数据样本的一个观测值，它给出了该观测值所属的聚类（类别赋值）。下面给出前面20个观测值的聚类赋值：

```
> head(clust,20)
[1] 1 2 2 2 1 2 3 3 2 3 1 3 2 3 2 1 2 1 1 3
```

把聚类号作为因子，就可以计算每个统计聚类的均值（参见方法6.5）。

```
> tapply(x, clust, mean)
        1          2          3
3.1899159 -2.6992682 0.2363639
```

这里R把数据很好地进行了聚类，每个聚类的均值看起来很不相同，一个聚类的均值为-2.7，一个聚类的均值为0.27，另一个聚类的均值为3.2（计算均值的顺序不必与原来组别的顺序一致）。提取出来的聚类的均值与原始的均值类似，但是不完全相同。三个聚类的箱线图可以说明其中的原因：

```
> plot(x ~ factor(means), main="Original Clusters", xlab="Cluster Mean")
> plot(x ~ factor(clust), main="Identified Clusters", xlab="Cluster Number")
```

图13-3的箱线图说明聚类算法把数据很好地分到了三个不相交的组别中。原始的组别有部分重合，聚类分析给出的组别没有重合。

图13-3说明了一维的数据，但函数dist同样可以用于多维的数据框或者矩阵数据。数据框或者矩阵的每一行是多维空间的一个观测值，函数dist可以计算这些观测值之间的距离。

另请参阅

本方法演示的是基于R基础发布版的聚类功能。其他的R包，例如mclust包中给出了其他可选的聚类方法。

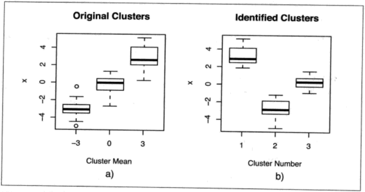


图13-3：聚类比较

13.7 预测二元变量（逻辑回归）

问题

需要执行逻辑回归或者建立一个能够预测二元事件出现的概率的回归模型。

解决方案

调用函数glm，并设置参数family=binomial来执行逻辑回归。其结果为一个模型对象：

```
> m <- glm(b ~ x1 + x2 + x3, family=binomial)
```

这里，b是一个有两个水平的因子（即取值为TRUE和FALSE，0和1）；x1、x2和x3是预测变量。

可以应用模型对象和预测函数predict来预测新数据的概率：

```
> dfrm <- data.frame(x1=value, x2=value, x3=value)
> predict(m, type="response", newdata=dfrm)
```

讨论

在建模中，二元结果的预测是很常见的。例如，某个治疗是否有效？价格是降低了还是提高了？队A，还是队B将赢得比赛？逻辑回归可以用来对这些情况进行建模。根据统计

学的精神，逻辑回归模型不是给出一个简单的“是”或者“不是”的答案。它给出两个可能结果中每一个可能发生的概率。

在调用函数predict时，我们设置参数type="response"，这时函数predict会返回一个概率值。否则，它将返回胜率的对数，而我没有发现后者有什么实际的用处。

在Faraway尚未出版的《Practical Regression and ANOVA Using R》一书中，给出了一个预测二元结果的例子：数据集为pima，如果病人检查为糖尿病阳性，则变量test的取值为TRUE。预测变量为舒张血压和身体重量指数（Body Mass Index，BMI）。Faraway在书中应用了线性回归，我们这里试着应用逻辑回归：

```
> data(pima, package="faraway")
> b <- factor(pima$test)
> m <- glm(b ~ diastolic + bmi, family=binomial, data=pima)
```

模型的汇总信息m说明预测变量diastolic和bmi的p值分别为0.805和0，因此我们说只有变量bmi是显著的。

```
> summary(m)

Call:
glm(formula = b ~ diastolic + bmi, family = binomial, data = pima)
Deviance Residuals:
    Min       1Q   Median       3Q      Max
-1.9128  -0.9180  -0.6848   1.2336   2.7417

Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -3.629553   0.468176  -7.753 9.01e-15 ***
diastolic   -0.001096   0.004432  -0.247    0.805
bmi          0.094130   0.012298   7.654 1.95e-14 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 993.48  on 767  degrees of freedom
Residual deviance: 920.65  on 765  degrees of freedom
AIC: 926.65

Number of Fisher Scoring iterations: 4
```

由于仅仅变量bmi是显著的，所以可以建立一个简化的模型：

```
> m.red <- glm(b ~ bmi, family=binomial, data=pima)
```

下面应用这个模型来计算一个中等体重指数（BMI值为32）的人糖尿病检查为阳性的概率。

```
> newdata <- data.frame(bmi=32.0)
> predict(m.red, type="response", newdata=newdata)
        1
0.3332689
```

按照模型，概率值大约为33.3%，同样可以计算BMI值第90个百分点上的人检查糖尿病为阳性的概率值为54.9%:

```
> newdata <- data.frame(bmi=quantile(pima$bmi, .90))
> predict(m.red, type="response", newdata=newdata)
      90%
0.5486215
```

另请参阅

应用逻辑回归时，需要根据对残差的解释来决定回归模型是否显著。这里建议在对回归下结论之前，参阅逻辑回归的相关书籍。

13.8 统计量的自助法

问题

有一个数据集和计算该数据集统计量的函数，现在需要估计该统计量的置信区间。

解决方案

应用R软件包boot中的函数boot来计算所需估计统计量的自助抽样：

```
> library(boot)
> bootfun <- function(data, indices) {
+       # . . . calculate statistic using data[indices] . . .
+       return(statistic)
+ }
> reps <- boot(data, bootfun, R=999)
```

这里的参数*data*是原始的数据集，它可以是向量的形式或者数据框的形式。计算统计量的函数bootfun有两个参数：data是原始的数据集，indices是一个整型向量，它用来从原始数据中选择自抽样样本。

然后，应用函数boot.ci来估计自助抽样的置信区间。

```
> boot.ci(reps, type=c("perc","bca"))
```

讨论

任何人都可以计算统计量，但是他们得到的是统计量的点估计。我们需要的是更高级的估计。那么什么是置信区间呢？对某些统计量，我们可以通过解析的方式来计算置信区间（Confidence Interval，CI）。以均值的置信区间为例，它可以应用函数t.test来计算。遗憾的是，这种计算方式只是一个特例，并不具有通用性。对大多数统计量而言，计算CI的数学公式或者太过复杂或者根本就没有这样的数学公式，因而没有计算置信区间的已知的解析形式的公式。即使在没有解析形式的置信区间的估计公式时，自助法也可以用来估计置信区间。自助法的工作原理是这样的：它假设有一个样本量为*N*的样本和一个计算统计量的函数。

1. 在原来的样本中随机地有放回抽取*N*个元素，这*N*个元素组成的集合称为自助抽样。
2. 应用统计量函数计算自助抽样的统计量值，这个统计量值称为自助复制。
3. 多次（一般几千次）重复进行第1步和第2步，得到相应的自助复制。
4. 从第3步得到的自助复制来计算置信区间。

最后一步看起来不可思议，事实是有多种方法来计算置信区间。一个简单的方法是采用自助复制的百分位数，例如取2.5百分位点和97.5百分位点来形成95%的置信区间。我是自助法的一个狂热爱好者。因为我每天工作中的统计量都是很古怪的，而我必须要知道它们的置信区间。解析形式的置信区间公式是绝对不知道的。这时自助法就可以给出一个很好的置信区间的估计。

让我们应用方法13.4中估计正交回归斜率的例子来说明自助法的应用。在方法13.4中，它给出了一个点估计，我们怎么能找到它的置信区间呢？

首先，我们用一个函数来计算斜率：

```
> stat <- function(data, indices) {
+          r <- prcomp( ~ x + y, data=data, subset=indices)
+          slope <- r$rotation[2,1] / r$rotation[1,1]
+          return(slope)
+ }
```

注意，该函数用参数indices定义的特定的索引来选择数据子集，然后在这个子集上计算斜率。

下一步，我们计算999个斜率的自助复制。在方法13.4中有两个向量*x*和*y*，这里把它们合并到一个数据框中。

```
> boot.data <- data.frame(x=x, y=y)
> reps <- boot(boot.data, stat, R=999)
```

这里选择重复的次数为999来进行初步计算，你可以选用更多的重复次数来观察结果是否有显著的改变。函数boot.ci从自助复制中计算置信区间。函数boot.ci实现了多种不同的算法，可以通过它的参数type设定需要应用的算法。对于选定的不同算法，boot.ci将返回相应的计算结果：

```
> boot.ci(reps, type=c("perc","bca"))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 999 bootstrap replicates

CALL :
boot.ci(boot.out = reps, type = c("perc", "bca"))

Intervals :
Level      Percentile            BCa
95%     ( 1.144, 2.277 ) ( 1.123, 2.270 )
Calculations and Intervals on Original Scale
```

这里，我通过设定参数type=c("perc","bca")，选择了两种计算自助法置信区间的算法。两种算法的计算结果在它们相应算法名称的下方。关于其他的算法，请参考函数的帮助文档。

注意，这两种算法给出的置信区间略微不同，它们分别是（1.144, 2.277)和(1.123, 2.270)。这令人不安，然而这是应用两种不同算法带来的不可避免的结果。我不知道如何确定哪种结果更好一些，如果选择一种最好的方法是至关重要的话，建议阅读相关文档，理解两种算法的不同之处。同时，我能给出的最好建议是保守地选择那个较宽的置信区间。本例中就是区间(1.123, 2.270)。

默认情况下，boot.ci给出95%的置信区间。你可以通过设置参数conf来改变置信水平为其他值，例如计算90%的置信区间:

```
> boot.ci(reps, type=c("perc","bca"), conf=0.90)
```

另请参阅

关于斜率的计算，参阅方法13.4。关于自助法的详细理论，可以参考Efron和Tibshirani所著的《An Introduction to Bootstrap》（Chapman & Hall/CRC）一书。

13.9 因子分析

问题

需要对数据集进行因子分析，找到变量间的公共因子。

解决方案

应用函数factanal，它的输入参数是原始的数据集和估计的因子的个数。

```
> factanal(data, factors=n)
```

输出包括n个因子，每个变量在因子上的载荷。输出结果中还有p值，一般情况下，如果p值小于0.05，表明因子的个数太少，而不能捕获原始数据集的足够信息；如果p值大于0.05，说明有足够的因子（或者过多）来捕获原始数据集的信息。

讨论

因子分析创建原始变量的线性组合（称为因子或者公共因子），因子则可以抽象表示原始变量的共同部分。如果N个变量是完全独立的，那么它们之间没有任何共同部分，将产生N个因子。一般情况下，如果原始变量有某种程度的共同性，那么可以得到少于N个的因子。

对于每个因子和原始变量，我们计算它们的相关系数，并称这些相关系数为因子载荷。有较大因子载荷的变量能够被因子较好地解释，变量的因子载荷的平方是该变量的总体方差能够被因子所解释的比例。

当少数的因子能够捕获原始变量的大部分的变差时，因子分析是有效的。它说明原始数据中有信息冗余，此时可以把密切相关的变量进行合并或者删除冗余的变量。因子分析的另一个巧妙应用是通过对因子的解释来找到变量间的相互关系。

如果两个变量在同一个因子上都有较大的载荷，就说明这两个变量之间有某种共性。什么共性呢？这里没有一个机械的答案。它需要你仔细理解数据和它们的含义。

因子分析有两个很关键的环节。一个是公共因子数目的选择。你可以先用主成分分析得到因子个数的初步估计。另外一个是因子本身的解释。

这里用股票价格，精确地说是股票价格的变化，来说明因子分析的应用。数据集包含12家公司的股票6个月期间股票价格的变化，每家公司和石油或者汽油业相关。这6家公司的业务类似，由相似的市场力量驱动，因此它们股票价格的变动有可能是一样的。问题是，需要多少因子来解释它们股票的变化？如果一个因子就可以，那么说明所有的股票都是一样的，一只股票和另一只股票都是一样好的。如果有多个因子，说明它们之间是有多样性的。

首先，应用主成分分析来分析价格的变化，数据集为diffs，它是表示价格变化的一个数据框。绘制主成分的结果，展示主成分捕获的变差的比例。

```
> plot(prcomp(diffs))
```

图13-4是主成分分析结果图，从图13-4中可知，前两个主成分捕获了大部分的变差，或许第三个也是需要的。根据这个结果，我们设定因子的个数为2进行初始的因子分析。

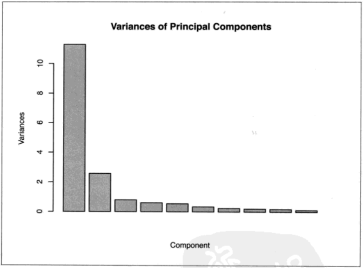


图13-4：价格变化的主成分

```
> factanal(diffs, factors=2)

Call:
factanal(x = diffs, factors = 2)

.
. (etc.)
.

Test of the hypothesis that 2 factors are sufficient.
The chi square statistic is 102.25 on 43 degrees of freedom.
The p-value is 9.83e-07
```

因为输出结果底部的p值接近于0（为9.83×10^{-7}），所以我们可以忽略大部分输出。小的p值说明两个因子是不够的，因此因子分析的结果不够好。需要更多的因子，下面设定3个因子进行因子分析：

```
> factanal(diffs, factors=3)
```

```
Call:
factanal(x = diffs, factors = 3)

Uniquenesses
  APC    BP   BRY   CVX   HES   MRO   NBL   OXY   SUN   TOT   VLO   XOM
0.005 0.509 0.299 0.162 0.150 0.207 0.321 0.265 0.098 0.185 0.257 0.155

Loadings:
    Factor1 Factor2 Factor3
APC 0.286   0.176   0.939
BP  0.212   0.150   0.651
BRY 0.628   0.474   0.286
CVX 0.819   0.320   0.253
HES 0.751   0.415   0.338
MRO 0.721   0.396   0.342
NBL 0.606   0.382   0.406
OXY 0.713   0.436   0.192
SUN 0.394   0.847   0.172
TOT 0.786   0.372   0.241
VLO 0.465   0.672   0.275
XOM 0.833   0.218   0.322

               Factor1 Factor2 Factor3
SS loadings      4.835   2.400   2.151
Proportion Var 0.403    0.200    0.179
Cumulative Var 0.403    0.603    0.782

Test of the hypothesis that 3 factors are sufficient.
The chi square statistic is 35.74 on 33 degrees of freedom.
The p-value is 0.341
```

这里的p值为0.341，大于0.05，它说明3个因子足够了，因此因子分析是有效的。输出中含有一个解释方差的表：

```
               Factor1 Factor2 Factor3
SS loadings      4.835   2.400   2.151
Proportion Var 0.403    0.200    0.179
Cumulative Var 0.403    0.603    0.782
```

这个表说明被每个因子解释的方差比例分别为0.403、0.200和0.179。它们解释的方差比例为0.782，没有解释的方差的比例只有$1 - 0.782 = 0.218$。

下一步，我们需要对因子给出解释，对因子的解释看起来更像是巫术，而不是科学。首先我们观察因子载荷：

```
Loadings:
    Factor1 Factor2 Factor3
APC 0.286   0.176   0.939
BP  0.212   0.150   0.651
BRY 0.628   0.474   0.286
CVX 0.819   0.320   0.253
HES 0.751   0.415   0.338
```

MRO	0.721	0.396	0.342
NBL	0.606	0.382	0.406
OXY	0.713	0.436	0.192
SUN	0.394	0.847	0.172
TOT	0.786	0.372	0.241
VLO	0.465	0.672	0.275
XOM	0.833	0.218	0.322

每一行用变量名来标记（即股票符号）：APC、BP、BRY等。第一个因子有多个大的因子载荷，表明这个因子解释了多只股票的方差。这种现象在因子分析中很常见，我们经常观察相关的变量和捕获它们的基本关系的第一个因子。在上面的例子中，我们分析的是股票，大多数股票和大的市场方向一起变化，这可以通过第一个因子来捕获。

第二个因子的解释更加微妙。注意变量SUN的载荷（0.847）和变量VLO的载荷（0.672）在第二个因子上较大，而其他股票的载荷都很小，它表明股票SUN和股票VLO有某种联系，也许它们在某个公共地区共同运营（比如，零售），因此它们一起变化。

第三个因子有两个主导地位的因子载荷，股票APC的载荷（0.939）和股票BP的载荷（0.651）。在本例中的取样期间，墨西哥湾地区有一个由于钻井平台爆炸而导致的可怕的环境灾难。它给未来石油勘探带来了阴影，在墨西哥湾地区，钻井平台由英国石油公司BP拥有，而阿纳达科石油公司APC是墨西哥湾地区石油勘探设备的主要供应商。当市场估计爆炸灾难对BP和APC的影响时，两只股票和谐地一起变化。

总之，这里的因子分析将股票分为三组：

- SUN和and VLO
- APC 和and BP
- 其他

因子分析是科学也是艺术。在应用因子分析之前，建议阅读一本好的多元统计分析的教材。

另请参阅

关于主成分分析，请参阅方法13.4。

第14章

时间序列分析

简介

随着数理金融的兴起和自动化证券交易的流行，时间序列分析已经成为一个热门学科。本章大部分的内容来源于金融、证券交易和投资组合管理的从业人员和研究人员。

在用R进行时间序列分析前，一个关键的问题是选择数据的表示方式（即对象类）。在像R这样的面向对象语言中，数据表示方式特别关键，因为数据表示方式的选择不仅影响数据的存储方式，而且它也决定了时间序列可以应用的函数（或方法）、处理、分析、输出和绘图等功能。例如，当我第一次使用R时，我只是把时间序列数据存储为向量，这似乎是很自然的。然而，我很快发现，R中没有任何可以应用于向量的功能强大的时间序列分析工具。于是我转到专门适用于时间序列分析的对象类，这是应用R进行时间序列分析的必经之路。

本章建议使用R的zoo包或者xts包来表示时间序列数据。这两个包是很通用的包，能满足大部分用户的时间序列分析需求。本章的所有分析都假设你使用这两种时间序列的表示方式。

警告： xts包是zoo包的超集，它实现了zoo包的所有功能，所以xts包能完成zoo包的一切功能，在本章中，除非另有说明，当zoo对象类能完成一项分析时，可以放心地应用xts对象类来完成分析。

与R基础包一起发布的有一个分析时间序列的类，即ts类。一般不建议用ts类进行时间序列分析，因为ts类本身的实现过于严格，并且和ts类相关的函数（或者方法）应用范围有限，功能亦不算强大。

日期和日期时间

时间序列的每一个观测值都有一个相联系的日期或者时间。在应用本章的对象类zoo和xts表达时间序列时，zoo对象类和xts对象类可供用户选择与该时间序列相联系的日期或日期时间。可以应用日期来表示日数据、周数据、月数据或者年数据。这里日期是得到观测值的日期（或者周、月、年）。可以用日期时间来表示一日内数据，它既有观测值发生的日期，也有发生的具体时间（例如，2012年3月3日18:10分。）。

由于本章将不断地提及日期或者日期时间，为了简单起见，下面将假设数据是日数据，采用一整天为时间单位。不过，用户可以自由采用日历日期下的任何时间标签。

另请参阅

R有许多有用的函数或者软件包来进行时间序列分析。可以在R网站“任务概览”（Task Review）栏目中找到这些相关的时间序列分析软件包（*http://cran.r-project.org/web/views/TimesSeries.html*）。

14.1 表示时间序列

问题

需要R的一个数据结构来表示时间序列。

解决方案

推荐应用zoo包和xts包。这两个包定义了适用于时间序列的数据结构，并且给出了许多用于时间序列分析的函数。假设x是一个向量或者数据框，dt是存储相应日期或日期时间的向量，可以用下列方式来创建一个zoo对象：

```
> library(zoo)
> ts <- zoo(x, dt)
```

创建一个xts对象：

```
> library(xts)
> ts <- xts(x, dt)
```

不同时间序列对象之间可以相互转换，它们由函数as.zoo和as.xts实现：

as.zoo(ts)
: 把ts转换为zoo对象。

as.xts(ts)
　　把ts转换为xts对象。

讨论

R至少有八种可以表示时间序列数据的数据结构，尽管我没有逐一尝试，但是zoo包和xts包是用于时间序列分析得很优秀的包，它们要大大好于我尝试过的其他时间序列分析包。

这两种表示数据的方法都假设你有两个向量：一个向量是数据的观测值，记为x，它为数值型数据；另外一个向量是相应的观测日期或时间，记为dt，它为日期或者日期时间。zoo包中的函数zoo把这两个向量组合为一个zoo对象：

```
> ts <- zoo(x, dt)
```

类似地，xts包中的xts函数把这两个向量组合为一个xts对象：

```
> ts <- xts(x, dt)
```

日期或者时间向量dt又称为索引，zoo包和xts包的索引不完全相同。

zoo
　　包的索引可以是任何有序值，例如Date对象、POSIXct对象、整数或者浮点数。

xts
　　包的索引必须是R支持的日期类或者时间类，包括Date对象、POSIXct对象和chron对象等最常用的对象，也可以是yearmon、yearqtr和dateTime对象。由于xts包实现了基于时间索引的强大操作，所以对索引的要求比zoo包要严格。

下面的例子创建了一个zoo对象，它包含IBM公司2010年最初5天的股票价格，它用Date对象作为索引。

```
> prices <- c(132.45, 130.85, 130.00, 129.55, 130.85)
> dates <- as.Date(c("2010-01-04", "2010-01-05", "2010-01-06",
+                    "2010-01-07","2010-01-08"))
> ibm.daily <- zoo(prices, dates)
> print(ibm.daily)
2010-01-04 2010-01-05 2010-01-06 2010-01-07 2010-01-08
    132.45     130.85     130.00     129.55     130.85
```

相反，下一个例子是IBM股票每秒的报价，索引所代表的时间是距离凌晨12:00的小时数。从早上9:30开始，每过1秒记录一个时间值，由于1秒大约为0.000 277 78小时，所以早上9:30为9.5小时，之后第1个1秒的时间为9.500 278小时，以此类推。代码如下所示。

```
> prices <- c(131.18, 131.20, 131.17, 131.15, 131.17)
> seconds <- c(9.5, 9.500278, 9.500556, 9.500833, 9.501111)
> ibm.sec <- zoo(prices, seconds)
> print(ibm.sec)
   9.5 9.5003 9.5006 9.5008 9.5011
131.18 131.20 131.17 131.15 131.17
```

上面两个例子都是单一时间序列，其中的时间序列数据来自一个向量。事实上，zoo对象和xts对象都可以处理多元时间序列，或者并行时间序列。只要把多个时间序列数据存储在一个数据框中，记为dfrm，然后执行和上面例子中类似的操作即可建立一个多元时间序列，例如：

```
> ts <- zoo(dfrm, dt)           # OR: ts <- xts(dfrm, dt)
```

第二个参数是相应于每个观测值的一个日期向量（或者日期时间向量）。这里只有一个日期向量与所有的时间序列数据相对应。换句话说，数据框中每一行的所有观测值的日期都是相同的。如果有的数据时间不匹配，请参照方法14.5。

如果时间序列数据存储在zoo对象或者xts对象中，那么可以通过函数coredata来提取时间序列数据。函数coredata返回一个简单向量或者矩阵：

```
> coredata(ibm.daily)
[1] 132.45 130.85 130.00 0.00 129.55
```

也可以通过函数index返回时间序列的日期或者时间：

```
> index(ibm.daily)
[1] "2010-01-04" "2010-01-05" "2010-01-06" "2010-01-07" "2010-01-08"
```

尽管xts包和zoo包十分类似，但xts包是经过速度优化过的，它特别适用于处理数据量大的时间序列。因此把其他时间序列数据表示方式转换为xts方式是明智的。

用zoo对象或者xts对象来表示时间序列数据的最大好处是可以直接应用专用的时间序列分析函数，例如输出、时序图、差分、合并、定期抽样和自动更新等。另外，zoo包有一个函数read.zoo，用来专门读取ASCII文件中的时间序列数据。

如前面所述，所有zoo对象可以完成的功能，都可以应用xts对象来完成。这两个包提供了许多有用的功能。

对于时间序列分析的专业人员，我强烈建议他们学习这两个R包的相关文档，这可以提高他们时间序列分析工作的效率。

另请参阅

在R的网站CRAN上有zoo包（*http://cran.r-project.org/web/packages/zoo/*）和xts包（*http://cran.r-project.org/web/packages/xts/*）的相关文档，例如参考手册、应用短文、快速参考卡等。如果这两个包已经安装，可以通过vignette函数来阅读它们的相关文档：

```
> vignette("zoo")
> vignette("xts")
```

另一个不错的分析时间序列的R包是timeSeries包，该包是数理金融项目——Rmetrics项目的一部分。

14.2 绘制时序图

问题

绘制一个或者多个时间序列的时序图。

解决方案

对于zoo对象或者xts对象，不管它们是单一时间序列还是多元时间序列，可以直接应用plot函数来绘制一个或多个时间序列的时序图。

如果有一个简单的时间序列观测值向量，记为v,可以应用函数plot(v,type="l")或者plot.ts(v)来绘制v的时序图。

讨论

泛型函数plot既适用于zoo对象也适用于xts对象。它可以绘制一个单独的时间序列，也可以绘制多元时间序列。如果是多元时间序列，那么可以选择在同一个坐标系中绘制多个序列，或者选择每个序列单独绘制。

假设ibm.infl是一个zoo对象，它含有两个时间序列，一个序列是IBM公司1970—1980年的股票价格，另一个序列是调整了通货膨胀因素后的同时期的IBM股票价格[注1]。如果用选项screens=1进行绘图，R会在同一个坐标系中同时绘制两个序列：

```
> plot(ibm.infl, screens=1)
```

注1： 这里的股票价格根据分红和配股进行了调整并正则化，使得起始价格为100，所以这里显示的数据和历史数据是不匹配的。

如果用选项screens=c(1,2)进行绘图,那么R会在两个图形中分别绘制这两个序列：

```
> plot(ibm.infl, screens=c(1,2))
```

默认情况下，plot函数不提供绘图的标题，它会给出默认的横轴标题和纵轴标题，但是默认情况给出的横轴标题和纵轴标题可能提供的信息不充分。可以定制plot函数的（标题）参数xlab、ylab和main的值，从而得到想要的横轴标题和纵轴标题以及绘图的标题。下列R代码将绘制出如图14-1和图14-2所示的图形。

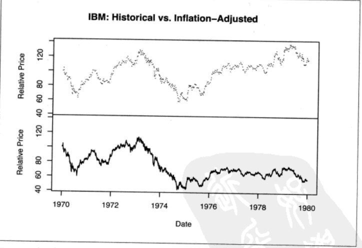


图14-1：独立绘制两个序列，参数screen=c(1, 2)

```
xlab="Date"
ylab="Relative Price"
main="IBM: Historical vs. Inflation-Adjusted"

lty=c("dotted", "solid")
ylim=range(coredata(ibm.infl))

# Plot the two time series in two plots
plot(ibm.infl, screens=c(1,2), lty=lty, main=main, xlab=xlab, ylab=ylab, ylim=ylim)

# Plot the two time series in one plot
plot(ibm.infl, screens=1, lty=lty, main=main, xlab=xlab, ylab=ylab)
```

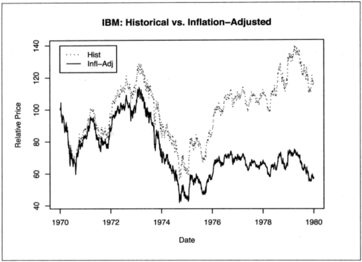


图14-2：同一坐标系绘制两个序列，参数screen=1

```
# Add a legend
legend(as.Date("1970-01-01"), 140,
       c("Hist","Infl-Adj"),
       lty=c("dotted", "solid"))
```

上述代码提供了两种线型（即参数lty）。所以，时序图中的两条线的类型不同，易于辨识。另外，对所有图形，它给出了统一的纵坐标值的范围（ylim），这样直观上便于比较。

这里同时给出了上述的两个图形，读者自己体验二者的不同之处。通过观察在同一坐标中的两个图，说明在20世纪70年代的通货膨胀中，IBM的股票被通胀大大“吞噬”了一口：调整通胀因素后，股票价格实际上下降了。

另请参阅

如果分析金融数据，quantmod包有专用的绘图函数，可以给出各种类型的漂亮图形。

14.3 提取最老的观测值或者最新的观测值

问题

需要查看最老的观测值或者最新的观测值。

解决方案

用函数head来查看最新的观测值：

```
> head(ts)
```

用函数tail来查看最老的观测值：

```
> tail(ts)
```

讨论

函数head和函数tail都是泛型函数，它适用于各种类型的数据，不管数据存储为简单的向量、zoo对象，或者xts对象。

假设有一个zoo对象，它存储了IBM股票过去几年的历史价格。你不能显示整个数据集，因为数据量大，屏幕会自动滚动。可以查看最初的几个观测值：

```
> head(ibm)
1970-01-02 1970-01-05 1970-01-06 1970-01-07 1970-01-08 1970-01-09
    364.75     368.25     368.50     368.75     369.50     369.00
```

也可以查看最后的几个观测值：

```
> tail(ibm)
1979-12-21 1979-12-24 1979-12-26 1979-12-27 1979-12-28 1979-12-31
     64.00      64.87      64.87      64.75      64.25      64.37
```

默认情况下，函数head和函数tail会各自显示最初始的和最新的6个观测值。可以通过设定函数的第2个参数来显示更多的观测值，例如：tail(ibm,20)。

xts包的函数first和函数last可以用日历日期来选择需要查看的观测值。可以用first和last来选定给定天数，或者给定周数，或者给定年数的观测值：

```
> first(as.xts(ibm), "3 weeks")
             [,1]
1970-01-02 364.75
1970-01-05 368.25
1970-01-06 368.50
1970-01-07 368.75
```

```
1970-01-08 369.50
1970-01-09 369.00
1970-01-12 367.75
1970-01-13 374.13
1970-01-14 373.75
1970-01-15 381.50
1970-01-16 369.75
> last(as.xts(ibm), "month")
           [,1]
1979-12-03 65.00
1979-12-04 65.37
1979-12-05 65.75
.
. (etc.)
.
1979-12-27 64.75
1979-12-28 64.25
1979-12-31 64.37
```

注意，这里先调用函数as.xts(ibm)，把zoo对象转换为xts对象，然后R调用xts的函数（first或者last）。还应注意，这里的“week”是用日历来定义的，而不是任何一个连续的7天。

另请参阅

可以通过应用help(first.xts)和help(last.xts)来查看函数first和函数last的详细的帮助文档。

14.4 选取时间序列的子集

问题

需要从时间序列中选择一个或者更多个特定的观测值。

解决方案

可以根据位置对zoo或者xts对象进行索引。根据对象含有一个或者多个时间序列来对时间序列对象应用一个或者两个下标：

ts[i]
: 从一个单一时间序列中选择第i个观测值。

ts[j,i]
: 选择多元时间序列的第j个时间序列的第i个观测值。

可以应用与时间序列索引类型相同的对象来选择数据。例如，假设时间序列的索引为Date对象，则可以应用：

```
> ts[as.Date("yyyy-mm-dd")]
```

可以通过日期序列进行索引：

```
> dates <- seq(startdate, enddate, increment)
> ts[dates]
```

应用函数window可以选择在一个时间范围内（开始日期和结束日期之间）的时间序列数据：

```
> window(ts, start=startdate, end=enddate)
```

讨论

回到前面IBM股票价格的例子：

```
> ibm.daily
2010-01-04 2010-01-05 2010-01-06 2010-01-07 2010-01-08
    132.45     130.85     130.00     129.55     130.85
```

与从向量中选择元素（方法2.9）一样，可以根据时间序列的位置来选择一个观测值：

```
> ibm.daily[2]
2010-01-05
    130.85
```

也可以用时间序列的位置来选择多个观测值：

```
> ibm.daily[2:4]
2010-01-05 2010-01-06 2010-01-07
    130.85     130.00     129.55
```

有时候，根据日期来选择数据会更实用。如果时间序列数据是以Date对象（或者以POSIXct对象）为索引的，则可以用Date对象（或者POSIXct对象）进行数据选择：

```
> ibm.daily[as.Date('2010-01-05')]
2010-01-05
    130.85
```

也可以用一个Date对象的向量来选择时间序列数据：

```
> dates <- seq(as.Date('2010-01-04'), as.Date('2010-01-08'), by=2)
> ibm.daily[dates]
2010-01-04 2010-01-06 2010-01-08
    132.45     130.00     130.85
```

也可以很方便地应用函数window选择在一个连续日期范围内的时间序列数据，例如：

```
> window(ibm.daily, start=as.Date('2010-01-05'), end=as.Date('2010-01-07'))
2010-01-05 2010-01-06 2010-01-07
    130.85     130.00     129.55
```

另请参阅

R的xts包提供了多种灵活的方式来对时间序列数据进行索引，请参阅xts包的帮助文档。

14.5 合并多个时间序列

问题

现有两个时间序列，需要把它们合并为一个时间序列。

解决方案

用zoo对象来表示时间序列，然后用函数merge来进行合并：

```
> merge(ts1, ts2)
```

讨论

当两个时间序列的时间标签不同时，对它们进行合并是很麻烦的。例如，现有两个时间序列，一个是IBM公司1970—1979年的股票每日价格，另一个是同一个时期的消费者价格指数（Consumer Price Index，CPI）。例如：

```
> ibm
1970-01-02 1970-01-05 1970-01-06 1970-01-07 1970-01-08 1970-01-09 1970-01-12
    364.75     368.25     368.50     368.75     369.50     369.00     367.75
.
. (etc.)
.
1979-12-21 1979-12-24 1979-12-26 1979-12-27 1979-12-28 1979-12-31
     64.00      64.87      64.87      64.75      64.25      64.37
> cpi
1970-01-01 1970-02-01 1970-03-01 1970-04-01 1970-05-01 1970-06-01 1970-07-01
      37.8       38.0       38.2       38.5       38.6       38.8       39.0
.
. (etc.)
.
1979-05-01 1979-06-01 1979-07-01 1979-08-01 1979-09-01 1979-10-01 1979-11-01
      71.5       72.3       73.1       73.8       74.6       75.2       75.9
1979-12-01
      76.7
```

很显然，这两个时间序列有不同的时间标签，一个是每日数据，另外一个是月数据。尤其糟糕的是，下载的CPI数据的时间标签是每个月的第一天，不管这一天是假期还是周末（例如元旦）。感谢功能强大的merge函数，它可以处理两个时间序列繁琐的时间标签的细节问题：

```
> merge(ibm,cpi)
              ibm  cpi
1970-01-01    NA  37.8
1970-01-02 364.75 NA
1970-01-05 368.25 NA
1970-01-06 368.50 NA
1970-01-07 368.75 NA
1970-01-08 369.50 NA
.
. (etc.)
.
```

默认情况下，函数merge会合并所有的日期（或者日期时间）：合并后的时间序列的时间标签是两个时间序列的所有日期，缺失的观测值由NA值来填充。可以应用zoo包中的函数na.locf，它用最新一期的观测值来替换时间序列的缺失值：

```
> na.locf(merge(ibm,cpi))
              ibm  cpi
1970-01-02 364.75 37.8
1970-01-05 368.25 37.8
1970-01-06 368.50 37.8
1970-01-07 368.75 37.8
1970-01-08 369.50 37.8
1970-01-09 369.00 37.8
.
. (etc.)
.
```

（这里locf代表“最后的观测值来进入”。）注意，上例中的NA值被替换了。同时，由于第一条观测值（1970-01-01）没有IBM股票价格，因此na.locf删除了该条记录。

也可以在na.locf函数的参数中应用选项all=FALSE来选取两个序列共有时间所对应的观测值，它仅仅保留两个序列都有的时间索引所对应的观测值。例如：

```
> merge(ibm,cpi,all=FALSE)
              ibm  cpi
1970-04-01 337.75 38.5
1970-05-01 296.75 38.6
1970-06-01 287.00 38.8
1970-07-01 254.00 39.0
1970-09-01 263.88 39.2
1970-10-01 297.00 39.4
.
. (etc.)
.
```

这里的输出是两个时间序列共有的时间所对应的观测值。

注意，合并后的序列是从1970年4月1日开始，而不是1970年1月1日。原因是1月1日、2月1日和3月1日这三天或者是节假日或者是周末，IBM股票没有观测值，所以和CPI数据的日期没有交集。如果需要更正开始日期，请参阅方法14.6。

14.6 缺失时间序列的填充

问题

时间序列有缺失值，需要填充这些缺失日期或时间以及它们相应的观测值数据。

解决方案

创建以缺失日期为索引的不含时间序列观测值的zoo对象，然后把原来的时间序列数据和该对象合并，取二者时间的并集：

```
> empty <- zoo(,dates)                  # 'dates' is vector of the missing dates
> merge(ts, empty, all=TRUE)
```

讨论

R的zoo包有一个方便地构建zoo对象的方法：可以构建一个仅含有时间而不含有观测值数据的zoo对象。可以用这个“假冒”的对象来执行对其他时间序列对象的填充。

假设你下载了1970—1979年的CPI月数据，时间标签是每月的第一天：

```
> cpi
1970-01-01 1970-02-01 1970-03-01 1970-04-01 1970-05-01 1970-06-01
      37.8       38.0       38.2       38.5       38.6       38.8
.
. (etc.)
.
1979-07-01 1979-08-01 1979-09-01 1979-10-01 1979-11-01 1979-12-01
      73.1       73.8       74.6       75.2       75.9       76.7
```

R软件仅仅知道除了每月的第一天外，其他日期没有观测值。而我们知道每月的CPI观测值是一样的。因此，我们首先建立一个没有观测数据，仅仅有1970—1979年10年的日期的零宽度的zoo对象：

```
> dates <- seq(from=as.Date("1970-01-01"), to=as.Date("1979-12-31"), by=1)
> empty <- zoo(,dates)
```

然后，我们把该零宽度的zoo对象和CPI时间序列数据进行合并，得到以下由NA值填充的数据集：

```
> filled.cpi <- merge(cpi, empty, all=TRUE)
> filled.cpi
1970-01-01 1970-01-02  1970-01-03  1970-01-04 1970-01-05 1970-01-06 1970-01-07
      37.8         NA          NA          NA         NA         NA         NA
1970-01-08 1970-01-09  1970-01-10  1970-01-11 1970-01-12 1970-01-13 1970-01-14
        NA         NA          NA          NA         NA         NA         NA
1970-01-15 1970-01-16  1970-01-17  1970-01-18 1970-01-19 1970-01-20 1970-01-21
        NA         NA          NA          NA         NA         NA         NA
1970-01-22 1970-01-23  1970-01-24  1970-01-25 1970-01-26 1970-01-27 1970-01-28
        NA         NA          NA          NA         NA         NA         NA
1970-01-29 1970-01-30 1970-01-31 1970-02-01 1970-02-02 1970-02-03 1970-02-04
        NA         NA         NA       38.0         NA         NA         NA
.
. (etc.)
.
```

合并后的新时间序列含有每一个日历日期，没有观测值的记录用NA填充。如果是需要的数据，此时就可以结束。通常的一个要求是用最近的观测值来替换每一个缺失值，这可以通过zoo包中的na.locf函数实现：

```
> filled.cpi <- na.locf(merge(cpi, empty, all=TRUE))
> filled.cpi
1970-01-01 1970-01-02 1970-01-03 1970-01-04 1970-01-05 1970-01-06 1970-01-07
      37.8       37.8       37.8       37.8       37.8       37.8       37.8
1970-01-08 1970-01-09 1970-01-10 1970-01-11 1970-01-12 1970-01-13 1970-01-14
      37.8       37.8       37.8       37.8       37.8       37.8       37.8
1970-01-15 1970-01-16 1970-01-17 1970-01-18 1970-01-19 1970-01-20 1970-01-21
      37.8       37.8       37.8       37.8       37.8       37.8       37.8
1970-01-22 1970-01-23 1970-01-24 1970-01-25 1970-01-26 1970-01-27 1970-01-28
      37.8       37.8       37.8       37.8       37.8       37.8       37.8
1970-01-29 1970-01-30 1970-01-31 1970-02-01 1970-02-02 1970-02-03 1970-02-04
      37.8       37.8       37.8       38.0       38.0       38.0       38.0
.
. (etc.)
.
```

1月1日的CPI观测值是37.8，它被应用到1月份的所有日期。2月1日的观测值为38.0，它用于填充2月份的所有日期的缺失值。以此类推，填充所有的缺失值。可以用这种方法来解决方法14.5中提到的问题（IBM日股票价格和月CPI在某些日期没有交集）。可以应用多种不同的方法来解决这个问题。其中一个方法是，填充IBM日股票数据，使其包含所有的CPI日期，然后与CPI数据合并取交集（注意，可以应用函数index(cpi)来得到CPI时间序列数据的所有日期）：

```
> filled.ibm <- na.locf(merge(ibm, zoo(,index(cpi))))
> merge(filled.ibm, cpi, all=FALSE)
           filled.ibm  cpi
```

```
1970-02-01    335.19 38.0
1970-03-01    340.25 38.2
1970-04-01    337.75 38.5
1970-05-01    296.75 38.6
1970-06-01    287.00 38.8
1970-07-01    254.00 39.0
.
. (etc.)
.
```

上面给出了月观测值。另外一种方法是与填充IBM数据类似地填充CPI数据，然后与IBM数据合并取交集,这样得到的是日观测数据：

```
> merge(ibm, filled.cpi, all=FALSE)
              ibm filled.cpi
1970-01-02    364.75 37.8
1970-01-05    368.25 37.8
1970-01-06    368.50 37.8
1970-01-07    368.75 37.8
1970-01-08    369.50 37.8
1970-01-09    369.00 37.8
.
. (etc.)
.
```

14.7 时间序列的滞后

问题

向前或者向后移动时间序列数据。

解决方案

应用函数`lag`。该函数有两个参数，第一个为原始时间序列，第二个为向前或者向后移动的期数，*k*：

```
> lag(ts, k)
```

如果第二个参数*k*为正数，则时间向前（明天的数据变为今天的数据）移动。如果*k*为负数，则实际上向后移动（昨天的数据变为今天的数据）。

讨论

回顾方法14.1中的IBM股票数据：

```
> ibm.daily
2010-01-04 2010-01-05 2010-01-06 2010-01-07 2010-01-08
    132.45     130.85     130.00     129.55     130.85
```

我们用k=+1，把时间序列向前移动一天：

```
> lag(ibm.daily, k=+1, na.pad=TRUE)
2010-01-04 2010-01-05 2010-01-06 2010-01-07 2010-01-08
    130.85     130.00     129.55     130.85         NA
```

如果设置na.pad=TRUE，则它用NA来填充缺失的数据；否则，如果设置na.pad = FALSE，则直接把有缺失值的数据删除，导致移动过后的时间序列数据变短。

可以用k=－1来把时间序列向后移动一天。由于数据后移导致开始记录的缺失，这里我们仍然设置na.pad=TRUE从而对缺失记录用NA填充。

```
> lag(ibm.daily, k=-1, na.pad=TRUE)
2010-01-04 2010-01-05 2010-01-06 2010-01-07 2010-01-08
        NA     132.45     130.85     130.00     129.55
```

这里*k*的取值看起来有点奇怪。

警告： 函数名为lag（中文为“滞后”），但是一个正的*k*值实际上产生超前数据而不是滞后数据。负的*k*值却得到滞后的数据，所以看起来怪怪的。也许函数名应该改为lead。

14.8 计算逐次差分

问题

给定一个时间序列，*x*，需要计算连续观测值之间的差分：$(x_2 - x_1), (x_3 - x_2), (x_4 - x_3), \ldots$

解决方案

应用diff函数：

```
> diff(x)
```

讨论

diff函数是一个泛型函数，它可以应用于简单的向量，也可以应用于zoo对象。优美的地方在于，zoo对象的差分仍然是一个zoo对象，并且有正确的日期。下面计算IBM股票两个连续交易日的差分：

```
> ibm.daily
2010-01-04 2010-01-05 2010-01-06 2010-01-07 2010-01-08
    132.45     130.85     130.00     129.55     130.85
> diff(ibm.daily)
2010-01-05 2010-01-06 2010-01-07 2010-01-08
     -1.60      -0.85      -0.45       1.30
```

日期为2010-01-05的差分值是相对前一天（即2010-01-04）的股票价格变化，这通常是大家需要的。由于R不能计算2010年1月4日相对前一个交易日的变化，所以差分后的序列比原来的序列要少一个观测值。

默认情况下，diff计算逐次差分，可以设置函数lag的参数，计算更宽距离的差分。假设没有CPI的月数据，现需要计算和去年同一个月的CPI值的差分，即相对去年同期的变化值，那么可以设定diff的第二个参数lag=12。

```
> head(cpi, 24)
1970-01-01 1970-02-01 1970-03-01 1970-04-01 1970-05-01 1970-06-01 1970-07-01
      37.8       38.0       38.2       38.5       38.6       38.8       39.0
1970-08-01 1970-09-01 1970-10-01 1970-11-01 1970-12-01 1971-01-01 1971-02-01
      39.0       39.2       39.4       39.6       39.8       39.8       39.9
1971-03-01 1971-04-01 1971-05-01 1971-06-01 1971-07-01 1971-08-01 1971-09-01
      40.0       40.1       40.3       40.6       40.7       40.8       40.8
1971-10-01 1971-11-01 1971-12-01
      40.9       40.9       41.1
> diff(cpi, lag=12)             # Compute year-over-year change
1971-01-01 1971-02-01 1971-03-01 1971-04-01 1971-05-01 1971-06-01 1971-07-01
       2.0        1.9        1.8        1.6        1.7        1.8        1.7
1971-08-01 1971-09-01 1971-10-01 1971-11-01 1971-12-01 1972-01-01 1972-02-01
       1.8        1.6        1.5        1.3        1.3        1.3        1.4
.
. (etc.)
.
```

14.9 时间序列相关的计算

问题

对时间序列数据进行算术运算或者对时间序列数据应用常见的函数。

解决方案

R对zoo对象有灵活的操作。可以应用算术运算符（+、-、*、/等），也可以应用常见的函数（例如sqrt、log等）来得到你期望的结果。

讨论

当对zoo对象进行算术运算时，R会根据日期自动将对象对齐，因此得到的结果有意义。假设需要计算IBM股票的百分比变化，我们需要用股票价格除每日价格的变化值，股票价格和每日价格的变化值这两个时间序列本来是不对齐的，它们有不同的开始日期，有不同的长度：

```
> ibm.daily
2010-01-04 2010-01-05 2010-01-06 2010-01-07 2010-01-08
    132.45     130.85     130.00     129.55     130.85
> diff(ibm.daily)
2010-01-05 2010-01-06 2010-01-07 2010-01-08
     -1.60      -0.85      -0.45       1.30
```

幸运的是，当用一个序列除以另外一个序列时，R会对齐这两个序列，返回一个zoo对象：

```
> diff(ibm.daily) / ibm.daily
  2010-01-05   2010-01-06   2010-01-07  2010-01-08
-0.012227742 -0.006538462 -0.003473562 0.009935040
```

可以把结果乘以100来得到百分比变化，结果存储为另一个zoo对象：

```
> 100 * (diff(ibm.daily) / ibm.daily)
2010-01-05 2010-01-06 2010-01-07 2010-01-08
-1.2227742 -0.6538462 -0.3473562  0.9935040
```

和上面的差分类似，在时间序列上也可以应用函数。例如，可以计算一个zoo对象的对数或者平方根，结果仍然是一个含有预存时间标签的zoo对象：

```
> log(ibm.daily)
2010-01-04 2010-01-05 2010-01-06 2010-01-07 2010-01-08
  4.886205   4.874052   4.867534   4.864067   4.874052
```

在投资管理中，常常需要计算价格的差分或者对数，这在R中是很容易实现的：

```
> diff(log(ibm.daily))
  2010-01-05   2010-01-06   2010-01-07  2010-01-08
-0.012153587 -0.006517179 -0.003467543 0.009984722
```

另请参阅

如果需要计算特殊情况下的逐次差分，请参见方法14.8。

14.10 计算移动平均

问题

计算时间序列的移动平均。

解决方案

应用zoo包的函数rollmean来计算*k*期移动平均。

```
> library(zoo)
> ma <- rollmean(ts, k)
```

这里的*ts*是时间序列数据，它以zoo对象方式存储，*k*是移动平均的期数。

对于大多数的金融应用，大部分是仅仅应用历史数据通过rollmean来计算移动均值，即只应用该日能够得到的数据。这可以通过设置参数align=right来实现；否则，函数rollmean会应用在该日尚不可得的未来数据计算均值：

```
> ma <- rollmean(ts, k, align="right")
```

讨论

交易员喜欢用移动平均来对波动的价格进行平滑，也称滚动均值。可以通过方法14.12中的技巧来计算滚动均值，它将函数mean和函数rollapply相结合，但是rollmean更快一些。

另外，rollmean 的优美之处还在于它返回一个zoo对象，所以该对象元素的日期是相对应的移动均值的日期。同样，由于是zoo对象，所以可以方便地把原始数据和移动均值进行合并。

由于rollmean需要*k*个完整的观测值才能计算移动均值，所以输出序列的初始数据点一般是缺失的。一般情况下，移动均值比原始输入的时间序列数据要短。如果需要输出序列和原来输入序列同等长度，可以设置参数na.pad=TRUE，这时rollmean会用NA值来填补缺失的初始数据点。

另请参阅

参见方法14.12来详细了解参数align。

这里的移动平均是简单移动平均。在R的quantmod包、TTR包和fTrading 包中有计算并绘制移动均值的许多函数，简单移动均值是其中之一。

14.11 在日历时间范围内应用函数

问题

给定一个时间序列，需要按照日历时间范围来对数据进行分组（例如周、月、年），然后把函数应用到每一组。

解决方案

R的xts包有根据天、周、月、季或者年等时间来处理时间序列的函数：

```
> apply.daily(ts, f)
> apply.weekly(ts, f)
> apply.monthly(ts, f)
> apply.quarterly(ts, f)
> apply.yearly(ts, f)
```

这里*ts*是一个时间序列，存储为xts对象。*f*是需要应用到每一天、每一周、每一月、每一季节，或者每一年的函数。

如果时间序列是zoo对象，首先把它转换为xts对象，然后即可应用这些函数。例如：

```
> apply.monthly(as.xts(ts), f)
```

讨论

按照日历时期（一个时间范围）来处理时间序列是很常见的。处理日历时期有时很繁琐，有时候会很奇怪。可以用本节中的函数来处理这一繁重的任务。假定有一个5年的IBM股票价格时间序列，数据存储为zoo对象：

```
> head(ibm)
1970-01-02 1970-01-05 1970-01-06 1970-01-07 1970-01-08 1970-01-09
    364.75     368.25     368.50     368.75     369.50     369.00
```

可以应用函数apply.monthly和mean一起来计算每月的平均价格：

```
> apply.monthly(as.xts(ibm), mean)
               [,1]
1970-01-30 359.8843
1970-02-27 346.8684
1970-03-31 327.4348
.
. (etc.)
.
1974-10-31 176.4791
1974-11-29 181.0570
1974-12-31 167.9233
```

另请参阅

注意，这里先把zoo对象用as.xts(ibm)转换为一个xts对象。如果一开始就把数据存储为xts对象，就不要进行这步转换了。

一个有用的应用是计算每月股票价格的波动率。波动率是日对数收益率的标准差。日对数收益率由下式计算：

```
diff(log(ibm))
```

然后逐月计算日对数收益率的标准差：

```
apply.monthly(as.xts(diff(log(ibm))), sd)
```

也可以根据调整的天数来计算年波动率：

```
sqrt(251) * apply.monthly(as.xts(diff(log(ibm))), sd)
```

可以把上述结果以图形方式表现出来，如图14-3所示，产生图形的代码如下：

```
> plot(sqrt(251) * apply.monthly(as.xts(diff(log(ibm))), sd),
+      main="IBM: Monthly Volatility",
+      ylab="Std dev, annualized")
```

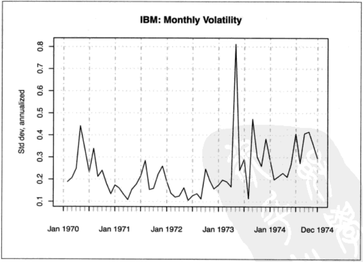


图14-3：IBM的月波动率

14.12 应用滚动函数

问题

需要在时间序列上以滚动的方式应用某个函数：在某个数据点计算函数值，然后移到下一个数据点，计算该函数的值，然后继续移到下一个点，以此类推。

解决方案

应用zoo包的rollapply函数。该函数的***width***参数定义在每一个点上需要函数（***f***）处理的时间序列（***ts***）中数据点的个数。

```
> library(zoo)
> rollapply(ts, width, f)
```

许多应用下，需要设置align="right"来避免函数*f*应用在计算点尚不可得的历史数据上：

```
> rollapply(ts, width, f, align="right")
```

讨论

rollapply函数会从时间序列中抽取一个时间窗口的数据子集，然后在该子集上应用函数，保存结果，然后，移动到下一个窗口，以此类推，直到完成整个输入数据。作为示例，考虑调用函数rollapply，这里设置width=21：

```
> rollapply(ts, 21, f)
```

函数rollapply会在时间序列ts的一个滑动窗口数据上重复调用函数f，如下所示：

1. f(ts[1:21])
2. f(ts[2:22])
3. f(ts[3:23])
4. *...(etc.)...*

这里函数f有一个向量值参数，函数rollapply保存函数f的返回值，然后把每个值加以时间标签并打包到一个zoo对象中。时间标签的选择取决于函数rollapply的参数align的取值：

align="right"：
　　时间标签取自最右边值的时间标签。

align="left"：
　　时间标签取自最左边值的时间标签。

align="center" (default)：
　　时间标签取自中间值的时间标签。

默认情况下，函数rollapply在连续数据点上重复应用函数f。也可以设置每隔*n*个数据

点来计算函数f。这里只要设置函数rollapply的参数by=*n*，完成一次函数计算后，将数据点向前移动*n*个。例如，当计算时间序列的标准差时，我们希望每个数据窗口是不重合的，可以设置参数by的值为窗口的大小：

```
> sds <- rollapply(ts, 30, sd, by=30, align="right")
```

rollapply需要一个完整的窗口应用到函数上，因此它通常跳过一些初始的数据。这样导致输出结果序列比原始输入序列短。可以通过设置参数na.pad=TRUE来强制两个序列等长，这时输出序列的初始部分会有NA值。

14.13 绘制自相关函数图

问题

绘制时间序列的自相关函数（Autocorrelation Function，ACF）图。

解决方案

应用acf函数：

```
> acf(ts)
```

讨论

自相关函数是揭示时间序列内部关系的一个重要工具。它是一组自相关系数 $\rho_k(k=1,2,3, ...)$的集合，这里ρ_k是时间序列所有相隔*k*步的点的自相关系数。

对这些系数进行可视化比列举这些值更为直观，所以函数acf绘制每个*k*值的自相关系数。图14-4是两个时间序列的自相关函数，其中一个时间序列有自相关，另一个没有自相关。图14-4中的水平虚线是相关系数是否显著的分界线：在虚线之上的是显著的（虚线的高度由时间序列数据的个数决定）。

显著的自相关是考虑应用自回归移动平均模型（Autoregressive Integrated Moving Average，ARIMA）来对时间序列建模的一个证据。从ACF，可以计算显著自相关的个数，据此来估计模型中移动平均（Moving Average，MA）系数的个数。图14-4a有7个显著的自相关系数，因此相应的ARIMA模型需要有7个MA系数（即MA(7)）。这里只是初步的估计，需要对模型进行拟合和诊断，然后验证该估计。

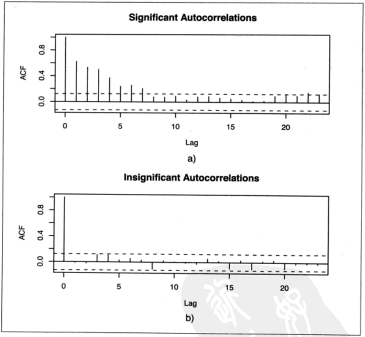


图14-4：两个自相关函数

14.14 检验时间序列的自相关

问题

需要检验时间序列中是否有自相关。

解决方案

应用函数Box.test，该函数实现了自相关系数的Box-Pierce检验：

```
> Box.test(ts)
```

函数输出一个p值，一般情况下，如果p值小于0.05，说明时间序列有显著的自相关；而p值大于0.05，则没有证据说明自相关存在。

讨论

绘制自相关函数会帮助获取数据中的信息。有时候，我们只需要知道数据是否有自相关性。而Box-Pierce检验可以直接提供相关性是否存在的信息。例如对方法14.13中绘制的两个序列应用Box-Pierce检验，得到的一个p值接近于0，另外一个p值为0.862。

```
> Box.test(ts1)

        Box-Pierce test

data:  ts1
X-squared = 96.9396, df = 1, p-value < 2.2e-16

> Box.test(ts2)

        Box-Pierce test

data:  ts2
X-squared = 0.0303, df = 1, p-value = 0.862
```

p值接近于0说明第一个时间序列有显著的自相关性。这里我们仅仅知道自相关性显著，而不能知道那一个相关系数显著。另一个p值为0.862，说明在第二个时间序列中没有检验到自相关。

函数 Box.test也可以进行Ljung-Box检验，在小的数据集上执行该检验较好。Ljung-Box检验计算的p值的解释和对Box-Pierce检验p值的解释是一样的。

```
> Box.test(ts, type="Ljung-Box")
```

另请参阅

参阅方法14.13中自相关函数绘制部分，它可以可视化地查看自相关函数。

14.15 绘制偏自相关函数

问题

绘制时间序列的偏自相关函数（Partial Autocorrelation Function，PACF）。

解决方案

应用函数pacf：

```
> pacf(ts)
```

讨论

偏自相关函数是另外一个揭示时间序列内部关系的工具。偏自相关函数的解释没有自相关函数的解释来的直观。该函数的具体数学定义请参阅统计教材。假设有两个随机变量X和Y，所谓偏自相关，直观讲就是在剔除了由于其他变量导致的随机变量X和Y的相关性后剩余的X与Y的自相关。对于时间序列而言，k步偏自相关函数是考虑相隔k步的数据点，在剔除了这k步之间的数据导致的相关性之后的相关性。

偏自相关函数可以帮助识别ARIMA模型中自回归（Autoregression，AR）系数的个数。图14-5给出了两个时间序列的PACF。其中一个有偏自相关，另一个没有。超出水平虚线的偏自相关函数是统计上显著的。有两个值，分别在$k=1$和$k=2$超出了虚线，因此初始的ARIMA模型将有两个AR系数，即AR(2)。与自相关类似，这只是一个初步的估计，必须通过模型拟合和诊断来验证模型的合理性。

另请参阅

参见方法14.13。

14.16 两个时间序列间的滞后相关性

问题

现有两个时间序列，二者之间是否有某种滞后相关性。

解决方案

应用函数ccf来绘制交叉相关函数，它将揭示滞后相关性：

```
> ccf(ts1, ts2)
```

讨论

交叉相关函数帮助揭示两个时间序列之间的滞后相关性。如果一个时间序列今天的值和另一个时间序列过去或者将来的值相关，则称二者有滞后相关性。

考虑商品价格和债券价格之间的关系。有些分析家相信二者是相关联的，因为商品价格变化是通货膨胀的一个晴雨表，而通货膨胀是债券定价的关键因素之一。我们可以发现它们之间的相关性吗？

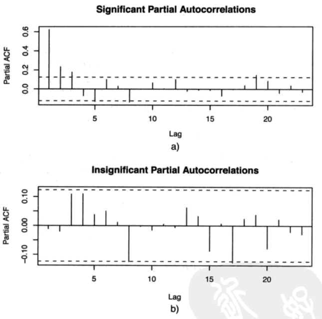


图14-5：两个偏自相关函数

图14-6是一个交叉相关函数图，一个序列为债券日价格变化，另一个序列为商品价格指数[注2]：

```
> ccf(bonds, cmdtys, main="Bonds vs. Commodities")
```

X轴是滞后期数，每个竖线是两个序列相应滞后期数的相关系数。如果竖线超出了两个水平虚线（向上或者向下），那么就有统计上显著的相关性。

注意，滞后期数为0的相关系数为−0.231，它是变量之间的简单相关系数：

```
> cor(bonds, cmdtys)
[1] -0.2309180
```

注2：　变量bonds是美国10年期国债期货的指数收益率，cmdtys是罗杰斯国际商品指数的指数回报率。数据的时间期间为：2007-01-03 日到2009-12-31。

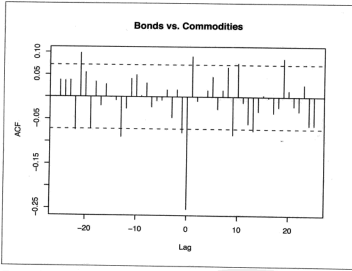


图14-6：交叉相关函数

注意，在滞后1期和−1期的相关性是统计显著的。很明显，因为今天的变化和明天的变化是相关的，所以债券价格和商品价格有“联动效应”。知道这种相关性有益于进行像市场分析和债券交易相关的短期预测。

14.17 剔除时间序列的趋势

问题

剔除时间序列含有的趋势。

解决方案

应用线性回归找出趋势成分，然后从原始时间序列中减去趋势成分。下面的代码给出了如何剔除zoo对象ts的趋势成分，结果存于变量detr中：

```
> m <- lm(coredata(ts) ~ index(ts))
> detr <- zoo(resid(m), index(ts))
```

讨论

有的时间序列含有趋势，它意味着随着时间的推移该序列逐渐向上倾斜或者向下倾斜。假设时间序列对象ts含有如图14-7a那样的趋势。

可以通过简单的两步来剔除趋势成分。首先，应用线性模型函数lm找出总的趋势。在线性模型中，用时间序列的时间作为x变量，观测值作为y变量：

```
> m <- lm(coredata(ts) ~ index(ts))
```

第二步，从原始的时间序列中减去线性模型发现的线性趋势。这一步的结果可以简单地取线性模型的残差，残差是原始序列和拟合的直线之间的差值：

$$r_i = y_i - \beta_1 x_i - \beta_0$$

这里r_i是第i项残差，β_1和β_0分别是线性模型的斜率和截距。可以应用函数resid来获取线性模型的残差，然后把它封装在zoo对象中即可：

```
> detr <- zoo(resid(m), index(ts))
```

注意，这里的时间索引和原始序列是一样的。当绘制detr时，它显然是没有趋势的，如图14-7b所示。

14.18 拟合ARIMA模型

问题

对时间序列拟合一个ARIMA模型。

解决方案

R软件包forecast的函数auto.arima可以自动选择合适的ARIMA模型的阶数并对数据进行模型拟合：

```
> library(forecast)
> auto.arima(x)
```

如果已经知道模型的阶数(p, d, q)的值，可以直接用函数ARIMA来拟合模型：

```
> arima(x, order=c(p,d,q))
```

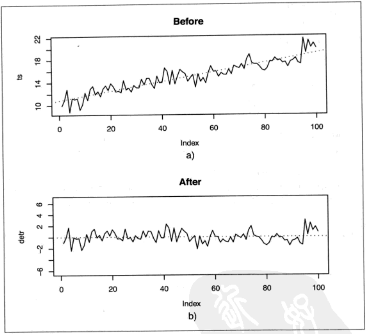


图14-7：剔除时间序列的趋势

讨论

建立一个ARIMA模型需要三个步骤：

1. 识别模型的阶数。
2. 对时间序列数据拟合模型，得到模型系数。
3. 进行模型诊断和模型验证。

模型的阶数由3个整数来表示，(p, d, q)，这里的p是自回归系数的个数，d是差分的阶数，q是移动平均系数的个数。当建立ARIMA模型时，一般不知道什么是合适的阶数。一般会应用函数`auto.arima`，让它来为我们选择合适的模型阶数，而不是手动繁琐地寻找p、d、q的最优组合：

```
> auto.arima(x)
Series: x
ARIMA(2,1,2)

Call: auto.arima(x = x)

Coefficients:
         ar1      ar2      ma1     ma2
      0.0451  -0.9183  -0.0066  0.6178
s.e.  0.0249   0.0252   0.0496  0.0517

sigma^2 estimated as 0.9877:  log likelihood = -708.45
AIC = 1426.9   AICc = 1427.02   BIC = 1447.98
```

在上例中，auto.arima找到了最优的阶数为(2, 1, 2)，即先对数据进行一阶差分，然后选择有两个自回归系数($p=2$)和两个移动平均系数的模型($q=2$)。

默认情况下，auto.arima限制p和q的范围分别为$0\leqslant p\leqslant 5$和$0\leqslant q\leqslant 5$。如果确信模型有少于5个的系数，那么可以设定参数max.p和max.q，这样可以限制继续寻找，使拟合速度更快。类似地，如果认为模型需要更多的系数，可以设置参数max.p和max.q来扩展搜索的范围。

如果已经知道ARIMA模型的阶数，函数arima可以快速地拟合时间序列数据：

```
> arima(x, order=c(2,1,2))
Series: x
ARIMA(2,1,2)

Call: arima(x = x,order=c(2,1,2))

Coefficients:
         ar1      ar2      ma1     ma2
      0.0451  -0.9183  -0.0066  0.6178
s.e.  0.0249   0.0252   0.0496  0.0517

sigma^2 estimated as 0.9877:  log likelihood = -708.45
AIC = 1426.9  AICc = 1427.02  BIC = 1447.98
```

输出结果和应用auto.arima的输出是完全一样的。你没有看到arima的执行速度更快。

从auto.arima和arima的输出都含有拟合的系数和系数的标准误差（s.e.）：

```
Coefficients:
          ar1      ar2      ma1     ma2
       0.0451  -0.9183  -0.0066  0.6178
s.e.   0.0249   0.0252   0.0496  0.0517
```

可以把ARIMA模型存储在一个对象中，然后应用函数confint找出系数的置信区间：

```
> m <- arima(x, order=c(2,1,2))
> confint(m)
```

```
          2.5 %      97.5 %
ar1 -0.003811699  0.09396993
ar2 -0.967741073 -0.86891013
ma1 -0.103861019  0.09059546
ma2  0.516499037  0.71903424
```

上述输出揭示了ARIMA模型的一个令人头痛的问题：不是所有的系数是必然显著的。参数ar1的置信区间为（-0.0038, +0.0940），它包含0，因此系数的真实值有可能为0，这时整个ar1就不需要了。同样，系数ma1的置信区间为（-0.0066, +0.0496）。由于ar2和ma2的置信区间不含有0，所以看起来是显著的。

如果模型含有不显著的系数，可以应用方法14.19来剔除它们。

函数auto.arima和函数arima含有拟合最优模型的一些有用的特性。例如，可以强制它们包含或者不包含趋势成分，参阅函数的帮助文档。

最后的告诫是：auto.arima的风险是它使建立ARIMA模型看起来简单。建立ARIMA模型不是一件简单的事情，它更像是艺术而不是科学。自动建立ARIMA模型仅仅是一个起点。在选定最终模型前，我建议读者参考有关ARIMA模型的好的教材。

另请参阅

有关ARIMA模型的诊断检验，请参阅方法14.20。

14.19 剔除ARIMA模型中不显著的系数

问题

需要剔除ARIMA模型中的一个或多个统计上不显著的系数。

解决方案

函数arima含有一个取值为向量的参数fixed。模型的每一个系数在该向量中都有相对应的元素。如果模型有常数项，那么该向量中亦有对应元素，向量每个元素的取值或者是NA或者是0。取值为NA，表示相应的模型系数需要保留；取值为0，表示剔除相应的模型系数。下面的例子是一个ARIMA(2, 1, 2)模型，其中第一个AR系数和第一个MA系数强制为0：

```
> arima(x, order=c(2,1,2), fixed=c(0,NA,0,NA))
```

讨论

方法14.8中的例子含有两个不显著的系数，ar1和ma1，通过查看它们的置信区间可以断定这两个系数是不显著的。

```
> m <- arima(x, order=c(2,1,2))
> confint(m)
           2.5 %      97.5 %
ar1 -0.003811699  0.09396993
ar2 -0.967741073 -0.86891013
ma1 -0.103861019  0.09059546
ma2  0.516499037  0.71903424
```

由于ar1和ma1的置信区间包含有0值，所以这两个系数的真实值可能为0，它们分别是模型的第一个参数和第三个参数。可以通过设置参数fixed的值，使第一个元素和第三个元素取值为0，因此可以剔除这两个系数：

```
> m <- arima(x, order=c(2,1,2), fixed=c(0,NA,0,NA))
> m
Series: x
ARIMA(2,1,2)

Call: arima(x = x, order = c(2, 1, 2), fixed = c(0, NA, 0, NA))

Coefficients:
     ar1     ar2 ma1    ma2
       0 -0.9082   0 0.5931
s.e.   0  0.0268   0 0.0522

sigma^2 estimated as 0.999: log likelihood = -711.27
AIC = 1428.54 AICc = 1428.66 BIC = 1449.62
Warning message:
In arima(x, order = c(2, 1, 2), fixed = c(0, NA, 0, NA)) :
  some AR parameters were fixed: setting transform.pars = FALSE
```

这里函数arima输出的警告信息是无关紧要的。函数arima的另一个参数是transform.pars，默认值为TRUE，在必要的情况下，函数arima会强制参数取值为FALSE，函数的输出中会就此给出警告信息。

注意，系数ar1和ma1现在取值为0。从置信区间可知，剩余的系数（ar2和ma2）仍然是显著的，因此我们得到了一个合理的模型：

```
> confint(m)
         2.5 %     97.5 %
ar1         NA         NA
ar2 -0.9606839 -0.8557283
ma1         NA         NA
ma2  0.4907792  0.6954606
```

14.20 对ARIMA模型进行诊断

问题

现在已经建立了一个ARIMA模型，需要就模型的有效性进行诊断检验。

解决方案

应用函数tsdiag。本例子把拟合的ARIMA模型对象存储在m中，然后对模型进行诊断：

```
> m <- arima(x, order=c(2,1,2), fixed=c(0,NA,0,NA), transform.pars=FALSE)
> tsdiag(m)
```

讨论

函数tsdiag是三个一组的图形。一个好模型产生的图形结果应该与图14-8给出的类似。那么什么样的图形是好的呢？

- 标准化的残差没有波动性聚集。
- 残差的自相关函数（ACF）没有显著的自相关性。
- Ljung-Box统计量的p值都比较大，表明残差没有任何模式。模型已经提取了数据中的所有信息，剩余的仅仅是“噪声”。

为了比较起见，图14-9是说明模型有问题的诊断图形。

- ACF值说明残差间有显著的自相关性。
- Ljung-Box统计量的p值比较小，表明残差具有某种模式。这说明数据中还有未被模型提取的信息。

以上都是基础的诊断，它们是模型诊断的起点。找一本好的有关ARIMA模型的参考书，在断定模型是一个好模型之前，需要进行书中推荐的诊断检验。其他的进行残差诊断的方法包括：

- 正态性检验
- Q-Q图
- 直方图
- 拟合值的散点图
- 时序图

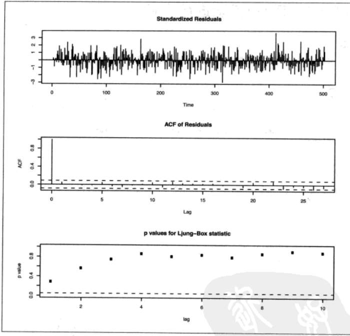


图14-8：诊断时间序列模型好的图形

14.21 用ARIMA模型进行预测

问题

对时间序列建立ARIMA模型，需要预测序列未来的观测值。

解决方案

保存ARIMA模型到一个对象，然后将函数predict应用于该对象。下例保存方法14.19中的ARIMA模型，然后预测未来的观测值：

```
> m <- arima(x, order=c(2,1,2), fixed=c(0,NA,0,NA), transform.pars=FALSE)
> predict(m)
```

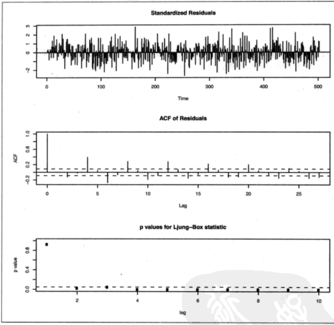


图14-9：诊断时间序列模型不好的图形

讨论

函数predict会根据模型计算未来的观测值和它的标准误差。返回值是一个有两个元素的列表，一个元素pred是预测值，另一个元素se是标准误差。继续应用解决方案中的例子，未来的预测值为-8.545，标准误差为0.9995：

```
> predict(m)
$pred
Time Series:
Start = 503
End = 503
Frequency = 1
[1] -8.545194

$se
Time Series:
```

```
Start = 503
End = 503
Frequency = 1
[1] 0.9995073
```

这里的pred和se都编码为时间序列，因此R会打印预测的数值，同时还有它们的时间序列属性。如果需要预测未来的观测值，可以设置参数n.ahead：

```
> predict(m, n.ahead=10)
$pred
Time Series:
Start = 503
End = 512
Frequency = 1
 [1] -8.545194 -9.365685 -9.365838 -8.620663 -8.620524 -9.297297 -9.297423
 [8] -8.682773 -8.682659 -9.240887

$se
Time Series:
Start = 503
End = 512
Frequency = 1
 [1] 0.9995073 1.4135168 1.5705650 1.7132774 1.9691066 2.1953229 2.3075363
 [8] 2.4145404 2.5935203 2.7609219
```

和上面一样，输出包含两个时间序列：一个是预测值，另外一个是标准误差。注意，每向后预测一个，标准误差会变大，这是因为越往后预测就会变得越不确定。

14.22 均值回归的检验

问题

需要知道所研究的时间序列是否有均值回归性质。

解决方案

检验均值回归的常用检验方法是增强的Dickey-Fuller检验（Augmented Dickey-Fuller，ADF），该检验由tseries包中的函数adf.test来实现。

```
> library(tseries)
> adf.test(coredata(ts))
```

函数adf.test输出检验的p值。习惯上，如果$p < 0.05$，则认为时间序列是均值回归的；否则，如果$p > 0.05$，则不认为时间序列有均值回归的性质。

讨论

这里讲的时间序列的均值回归性质，是指时间序列倾向于回到该序列的长期均值。虽然时间序列会偏离它的长期均值，但是最后时间序列将返回到它的长期均值。如果一个时间序列不是均值回归的，那么它偏离了长期均值之后可能永远不会回到长期均值。

图14-10中上图的时间序列看起来向下偏离并且不会返回到其长期均值。下面的adf.test给出的 p 值也确认了该时间序列不是均值回归的。

```
> adf.test(coredata(ts1))

        Augmented Dickey-Fuller Test

data: ts1
Dickey-Fuller = -2.1217, Lag order = 6, p-value = 0.5246
alternative hypothesis: stationary
```

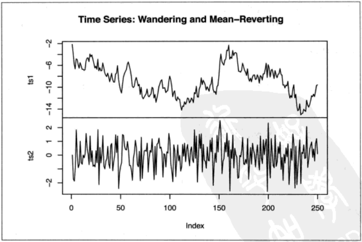


图14-10：两个时间序列：上图偏离均值；下图均值回归

图14-10中下图的时间序列看起来围绕它的长期均值上下波动。下面的adf.test给出的极小 p 值也确认了该时间序列是均值回归的。

```
> adf.test(coredata(ts2))

        Augmented Dickey-Fuller Test
```

```
data: ts2
Dickey-Fuller = -7.1752, Lag order = 6, p-value = 0.01
alternative hypothesis: stationary
```

在执行ADF检验之前，函数adf.test首先对数据进行平滑。首先它自动对数据移除趋势，然后对数据进行重新中心化，使得重新中心化数据的均值为0。如果应用不希望移除数据的趋势或者不希望重新中心化，可以选用fUnitRoots包中的函数adfTest来进行ADF检验[注3]。

```
> library(fUnitRoots)
> adfTest(coredata(ts1), type="nc")
```

函数adfTest中的参数type="nc"，告诉函数不要对数据进行趋势移除也不要进行中心化。如果参数设置为type="c"，函数会对数据进行重新中心化，但是不移除趋势。

函数adf.test和函数adfTest允许通过参数来设定滞后期数，这样可以明确地指定函数所计算的统计量。尽管这两个函数都会提供合理的默认滞后期数，但谨慎的用户还是应该学习参考资料中ADF检验的详细描述，然后确定适宜于他们的时间序列的滞后期数。

另请参阅

另外两个R包（即urca和CADFtest）也实现了单位根检验，它们可以用于均值回归的检验。需要注意的是，在比较多个不同包的检验结果时，每个包可能有稍微不同的假设，这将导致它们得到的结果有细微的区别。

14.23 时间序列的平滑

问题

需要对一个有噪声的时间序列进行平滑以消除噪声。

解决方案

R软件包KernSmooth含有平滑函数。应用函数dpill来选择初始带宽参数，然后应用函数locploy来对数据进行平滑：

```
library(KernSmooth)
```

注3：　这里要注意函数adf.test和函数adfTest的拼写差别。不幸的是，这种细微的差别在R中是很常见的。它们常见于R的几个不同包中，这些包实现了同一个函数功能但是却不完全相同。

```
gridsize <- length(y)
bw <- dpill(t, y, gridsize=gridsize)
lp <- locpoly(x=t, y=y, bandwidth=bw, gridsize=gridsize)
smooth <- lp$y
```

这里，t是时间变量，y是时间序列。

讨论

R软件包KernSmooth是一个标准的R发布版，其中的函数locpoly会在每一个数据点构建一个可以拟合周围点的多项式，称为局部多项式。这些局部多项式连在一起构成了原始时间序列数据的一个平滑的版本。

平滑算法需要一个带宽（bandwidth）参数，它控制平滑的程度。小的带宽意味着平滑程度低，这时平滑结果更接近原始的数据；较大的带宽意味着更加平滑，平滑结果中就含有较少的噪声。这里的难点是如何选择合适的带宽，既不过大，也不太小。

幸运的是，KernSmooth包中有一个函数dpill，它可以较好地估计出合适的带宽。我的建议是先从函数dpill给出的带宽值开始，然后以该值为基础试验较大的值和较小的值。这里没有灵丹妙药，你需要自己决定适合你应用的合适的平滑程度。

图14-11给出了平滑的一个例子。实线是一个由简单正弦曲线和正态分布的"噪声"之和构成的时间序列：

```
t <- seq(from=-10,to=10,length.out=201)
noise <- rnorm(201)
y <- sin(t) + noise
```

函数dpill和函数locpoly需要一个网格大小（grdisize）参数，它设定构造局部多项式所用数据点的个数，一般情况下设置该参数的值为总的数据点的个数，这样可以得到较好的分辨率，得到的结果是一个很平滑的时间序列。如果数据集很大，或者你需要较粗粒度的分辨率，则可以选择一个较小的网格大小值：

```
library(KernSmooth)
gridsize <- length(y)
bw <- dpill(t, y, gridsize=gridsize)
```

函数locpoly进行平滑操作，并返回一个列表，列表的元素y是平滑后的数据：

```
lp <- locpoly(x=t, y=y, bandwidth=bw, gridsize=gridsize)
smooth <- lp$y
```

在图14-11中，虚线是平滑后的数据，这幅图形说明函数locpoly很好地抽取出了原来的正弦曲线。

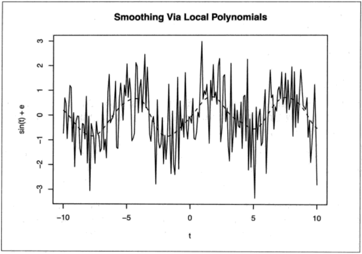


图14-11：平滑时间序列

另请参阅

R基础包中的函数 ksmooth、lowess和HoltWinters都可以进行平滑。R包expsmooth实现了指数平滑。